TURING 图灵程序设计丛书

MANNING

jQuery in Action Second Edition

jQuery实战（第2版）



[美] Bear Bibeault
Yehuda Katz 著
三生石上 译

人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。



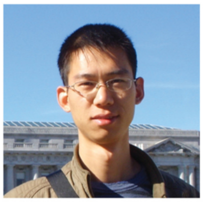
Bear Bibeault

著名Web技术专家，有30多年编程经验，也是技术社区JavaRanch的核心人物之一。除本书外，他还和其他世界级Web专家联袂打造了巨著 《Ajax实战：实例详解》和《Ajax实战：Prototype与Scriptaculous篇》（均由人民邮电出版社出版）。



Yehuda Katz

著名Web技术专家，jQuery开发团队的核心成员，Merb等开源项目的贡献者。他还维护着热门网站VisualjQuery.com。



三生石上

毕业于中国科学技术大学，曾在思科网迅任前端高级工程师，著名开源框架ExtAspNet创始人。

TURING 图灵程序设计丛书

jQuery in Action Second Edition

jQuery实战（第2版）



[美] Bear Bibeault
Yehuda Katz 著
三生石上 译

人民邮电出版社
北京

图书在版编目（ＣＩＰ）数据

jQuery实战 : 第2版 / (美) 比伯奥特
(Bibeault,B.）, (美) 卡茨 (Katz,Y.) 著 ; 三生石上
译. -- 北京 : 人民邮电出版社, 2012.3
(图灵程序设计丛书)
书名原文: jQuery in Action Second Edition
ISBN 978-7-115-27457-1

Ⅰ. ①j… Ⅱ. ①比… ②卡… ③三… Ⅲ. ①
JAVA语言—程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第023600号

内 容 提 要

jQuery 是目前最受欢迎的 JavaScript/Ajax 库之一，能用最少的代码实现最多的功能。本书全面介绍 jQuery 知识，展示如何遍历 HTML 文档、处理事件、执行动画、给网页添加 Ajax 以及 jQuery UI 。书中紧紧地围绕“用实际的示例来解释每一个新概念”这一宗旨，生动描述了 jQuery 如何与其他工具和框架交互以及如何生成 jQuery 插件。

本书适合各层次 Web 开发人员。

图灵程序设计丛书

jQuery实战（第2版）

◆ 著　[美] Bear Bibeault　Yehuda Katz
译　三生石上
责任编辑　朱　巍
执行编辑　丁晓昀

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：26
字数：617千字　2012年 3 月第 1 版
印数：1－4 000 册　2012年 3 月北京第 1 次印刷

著作权合同登记号　图字：01-2010-6924号

ISBN 978-7-115-27457-1

定价：69.00元

读者服务热线：(010)51095186转604　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版权声明

对第1版的赞誉

这是一本杰出的著作，是Manning出版社“In Action”系列的成功延续。它浅显易懂且示例丰富。实验室页面是探索jQuery库的绝佳方式，它应该成为所有Web开发者工具箱中的重要组成部分。给予五星评价！

——David Sills，JavaLobby，Dzone.com

此书既可以作为初学jQuery时的基础教材，也可以作为精通jQuery之后的参考手册。

——David Hayden，微软MVP C#，Codebetter.com

对于jQuery的新手来说，此书是很好的初级读本，它包含了很多框架的常见用法……示例贯穿全书并相互关联，使论述更有条理。代码片段很明显，文字清晰易懂。

——Grant Palin，Blogger.com

感谢本书的作者Bear Bibeault 和Yehuda Katz，他们的做法值得效仿。你可以采取从头到尾的方式学习这本综合性著作，当然你也可称之为操作手册。对jQuery已经有一定涉猎并需要最佳实践验证的开发者来说，此书也是很棒的参考手册。

——Matthew McCullough，美国科罗拉州丹佛开源用户组

本书技术覆盖全面，示例代码使用广泛，文字浅显易懂，对任何Web开发者来说都是非常有价值的资料，能帮助他们最大限度地利用JavaScript的强大功能，是所有对学习jQuery感兴趣的人的必读之作。

——Michael J. Ross，
Slashdot贡献者，Web开发人员

我强烈推荐此书，无论新手还是高级开发者，只要想深入学习JavaScript、编写最好最优雅的代码，又不想被JavaScript的传统编程方式所困扰，都可以从中获益。

——Val的博客

*jQuery in Action*对崭露头角的jQuery库（客户端JavaScript）进行了全面深入的研究。

——www.DZone.com

第 1 版 序

一切为了简单。当Web开发者想要编写几个简单交互行为的时候，为什么非得把代码写得冗长、复杂，像书一样长篇大论呢？复杂性从来就不是Web应用开发的必要条件。

在着手创建jQuery的时候，我就决定把重点放在小巧而简单的代码上，它们要服务于Web开发者日常处理的实际应用。读了本书之后，我感到很高兴，因为本书出色地体现了jQuery库的原则。

本书注重以简洁、精炼的形式展示实用的代码，对于想熟悉该库的人来说，它是理想的学习资源。

Bear和Yehuda对该库内部工作机制的细节给予了极大的关注，这是本书最让我满意的地方。他们不遗余力地研究和推广jQuery API，我几乎每天都能收到来自他们的电子邮件或即时消息，请求解释、报告新发现的程序缺陷以及对库的改进建议。因此请放心，摆在你面前的这本关于jQuery库的书，是经过作者深思熟虑并深入研究后完成的著作。

本书还清晰地阐述了jQuery插件以及插件开发背后的策略和理论，这也让我感到很惊喜。插件体系结构的运用使得jQuery非常简单。这个体系结构提供了许多有文档描述的扩展点，插件可以基于此添加扩展功能。一般来说，那些功能虽然有用却不够通用，所以没有被纳入jQuery——因此插件体系结构十分必要。本书讨论的一些插件，例如Forms（表单）、Dimension（尺寸）和LiveQuery（实时查询）插件，已被广泛采用，其原因非常明显：它们的创建、文档编写和维护都是专家级的。请特别关注怎样使用和构建插件，因为插件的运用是jQuery开发的基石。

有如此优秀的资源加以佐助，jQuery项目一定会继续成长并取得更大的成功。在你探索jQuery的征途中，它将是你的得力助手。

John Resig
jQuery的创建者

第1版前言

本书两位作者，一位是头发已经斑白的老手，早在FORTRAN盛行的年代他就专注于编程，而另一位是狂热的领域专家，聪明过人，一旦脱离因特网，他就会对外面的世界茫然若失。背景如此迥异，是什么促使他们走到一起合作项目呢？

答案很明显，jQuery。

一个非常有用的客户端工具使我们开始合作，但两人的经历却又截然不同。

我（Bear）第一次听说jQuery是在写*Ajax in Practice*[①]的时候。图书出版流程的后期是暴风骤雨般的编校阶段，文字编辑审阅全文以确保语法正确和语句清晰，技术编辑保证技术无误。至少对我来说，这是写书过程中压力最大、最令人抓狂的阶段，我最不想听到的就是“你真应该增加一节全新的内容”。

我在*Ajax in Practice*中的一章论述了一些支持Ajax的客户端库，我对其中一个库（Prototype）相当熟悉，而对其他库（Dojo工具箱和DWR）只好一笔带过。

就在我忙于应付众多任务的时候（保住工作，发展副业，处理家庭琐事），技术编辑Valentin Crettaz不经意地爆出了一句：“你为什么不写一节关于jQuery的内容？”

“j什么？”我问。

他立即发表长篇大论，描述了这个全新的库是如何奇妙，真应该与当前支持Ajax的其他客户端库一起进行评测。于是我问了一些人：“有没有听说过‘jQwery’库？”

我收到了很多积极回应，这些回应都很热情，一致认为jQuery非常出色。在一个细雨绵绵的周日下午，我花了大约4小时在jQuery网站阅读文档，并编写小测试程序，摸索jQuery的行事方式。然后我匆匆写出一节新的内容，并发给技术编辑看看是否达到要求。

这节内容受到热情赞扬，随后我们继续推进并最终完成了*Ajax in Practice*一书的编辑工作（关于jQuery的这节内容后来还发表在*Dr. Dobb's Journal*的网站上）。

当尘埃落定，对jQuery的狂热已经让一颗倔强的小种子在我内心深处生根发芽。我很喜欢之前在匆忙中研究jQuery所学到的知识，于是就开始进一步学习。我开始在Web项目中使用jQuery。我很喜欢我看到的效果。我开始替换以前项目中的老代码，看如何使用jQuery简化页面。我变得对它爱不释手。

这个新发现让我满腔热情并想与他人分享，我冲动地向Manning出版社递交了出版本书的提

① 中文版《Ajax实战：实例详解》，人民邮电出版社，2008。——编者注

议。显然，我那时一定很有说服力。（我让引起这一切麻烦的技术编辑Valentin继续做本书的技术编辑，作为对他的惩罚。我打赌这对他来说是深刻的教训。）

就在那时，编辑Mike Stephens问我："这个项目和Yehuda Katz合作，你觉得怎么样？"

"Yehenta是谁呀？"我问道。

*　　　　*　　　　*

Yehuda参与这个项目的经历和我大相径庭。早在jQuery连版本号都没有的时候，他就参与进去了。在无意中发现Selectable插件之后，他就对jQuery核心库产生了兴趣。他对于（那时）缺少在线文档多少有点失望，于是开始更新维基网页，并且建立了Visual jQuery网站（visualjquery.com）。

之后不久，他牵头完善在线文档，为jQuery做贡献，管理插件体系结构和生态系统，同时他还向Ruby社区推广jQuery。

然后有一天，Manning出版社给他打来电话（一个朋友把他的名字告知了出版社），问他是否有兴趣和一个名叫Bear的家伙合作，出一本关于jQuery的书……

*　　　　*　　　　*

虽然我们的背景、经历、专长迥异，参与项目的方式也不同，但并不妨碍我们组成一个很棒的团队，并且在合作的过程中收获了许多快乐。即便是地理上的差距（我在德克萨斯州的中心，而Yehuda在加利福尼亚州的海边），也不足以形成障碍。这得感谢电子邮件和即时消息带来的便利！

把我们的知识和才华凝聚成一本很棒且有深度的好书，这正是我们所期望的。希望你阅读愉快，恰如我们创作时那样。

但同时也要保持清醒的头脑。

前　　言

距本书第1版出版才过去了两年，有必要这么快对其进行更新吗？

当然！

与稳定的服务器端语言（例如Java）相比，Web客户端技术的更新要快速得多。jQuery不是东拼西凑的技术，而是走在趋势最前沿的技术！

jQuery团队每年都会发布一个主要的新版本（最近力争每年1月份发布）。除此之外，在一年之中还会有一些小版本更新。这意味着自本书第1版出版以来，已经有很多小版本更新以及两个基于jQuery 1.2的主要版本，也就是jQuery 1.3和1.4版本。①

随着每个主要版本的发布，jQuery的功能都得到了极大扩展和增强。无论是从自定义事件，事件的命名空间，函数和特效队列，还是从大量新增的实用方法和函数来看，在第1版大获成功之后，jQuery的能力范围已经得到了显著扩大。

这还不包括jQuery UI！早在两年前的初期阶段，本书的第1版只用了其中一章的几节来介绍jQuery UI。从那以后，jQuery UI逐渐流行并发展成熟，在这个版本中，jQuery UI已是重要的一部分，总共有3章。

因此第2版的发行顺理成章，它包含了jQuery和jQuery UI在过去两年中所取得的发展成果。

第 2 版有什么新的内容？

当我们决定继续编写本书的第2版时，我记得有人曾跟我说过，“这应该是小菜一碟。你只需要对第1版进行一些更新就可以了。”

他们完全错了！事实上，完成第2版所花的时间比编写第1版还要多。我们不想掉入这个“理所当然”的陷阱，仅仅在某些地方增加一些更新就万事大吉了。我们想让这一版不仅是第1版的重温，而且还应该包含更多的内容。

比较一下第1版和第2版的目录就可以知道，从第1章到第8章的目录结构并没有太大变化。但这也是两个版本仅有的相似之处了。

第2版不是对第1版的简单修补。文本中每一个段落，示例中的每一行代码，都经过了仔细的检验。书中不仅包含了jQuery 1.2~1.4版本新增和修改的内容，而且每章的内容和示例代码也已更

① 在编辑本书中文版时，jQuery目前的最新版本为1.7.1。——编者注

新，以反映当前页面脚本和jQuery用法的最佳实践。总之，作为一个团队，在使用jQuery编写高度交互的页面脚本方面，我们又拥有了另外两年的实践经验。

我们检查了所有示例，更新了一些示例以便更好地展示如何在实践中使用jQuery 1.4，把一些示例替换成更适合展示讨论内容概念的示例。例如，第1版的读者可能还记得，第4章结尾处用来展示jQuery事件处理的综合示例Bamboo Grille。经过尝试，我们还是没能重构此示例来展示最新的jQuery事件处理概念，比如“live”和自定义事件。因此，我们用示例DVD Ambassador代替了这个示例，后者能更好地演示高级事件处理概念。

本书的第二部分聚焦jQuery UI，这是全新的内容，全面覆盖了自第1版出版以来jQuery UI的更新。

我们估算了一下，将本书第一部分新增、替换和更新的内容，以及全新的第二部分计算在内，本书至少有50%是全新的内容。余下的50%内容也经过了全面检查以确保其内容是最新的，能反映最新的最佳实践。

这已经不是“小菜一碟”了！

致　　谢

电影结束时银幕上滚动的、无穷无尽的致谢名单是否曾让你感到惊讶甚至困惑？真的需要那么多人去制作一部电影吗？

与此类似，可能很多人会为需要那么多人参与一本书的出版而感到惊讶，但事实就是如此。才华横溢的贡献者通力合作，最终成果就是你手上拿着的这本书（或是你在屏幕上阅读的电子书）。

Manning出版社的全体员工一直孜孜不倦地和我们一起工作，以确保本书达到预期的高水平，感谢他们所做的努力。没有他们，就没有本书。致谢名单不仅包括出版人Marjan Bace、编辑Mike Stephens，也包括以下贡献者：Lianna Wlasiuk、Karen Tegtmayer、Andy Carroll、Deepak Vohra、Barbara Mirecki、Megan Yockey、Dottie Marsico、Mary Piergies、Gabriel Dobrescu和Steven Hong。

对审稿人我们更是感激不尽，他们帮忙确定本书的最终形式，从发现简单的打字错误，到更正术语和代码错误，甚至帮助组织本书章目。每个审稿环节都极大提升了最终产品的质量。感谢抽出时间审阅本书的各位朋友：Tony Niemann、Scott Sauyet、Rich Freedman、Philip Hallstrom、Michael Smolyak、Marion Sturtevant、Jonas Bandi、Jay Blanchard、Nikander Bruggeman、Margriet Bruggeman、Greg Donald、Frank Wang、Curtis Miller、Christopher Haupt、Cheryl Jerozal、Charles E. Logston、Andrew Siemer、Eric Raymond、Christian Marquardt、Robby O'Connor、Marc Gravell、Andrew Grothe、Anil Radhakrishna、Daniel Bretoi和Massimo Perga。

特别感谢本书的技术编辑 Valtentin Crettaz。他除了在各种条件下检查每段示例代码，还为本书做出了宝贵的贡献，包括确保文字的技术正确性，查找原本丢失的信息，紧跟库的更新步伐，以确保本书中使用的是最新和最准确的jQuery和jQuery UI，甚至当需要服务器端代码时，他还提供了PHP版的示例。

Bear Bibeault

本书是我的第四本著作，需要感谢很多人，再次感谢javaranch.com网站的全体会员和职员。如果起初我没有加入JavaRanch，就不会有机会去写书，因此诚挚地感谢Paul Wheaton和Kathy Sierra，是他们启动了整件事情，同时感谢给予我鼓励和支持的会员，包括（但不限于）Eric Pascarello、Ben Souther、Ernest Friedman Hill、Mark Herschberg、Andrew Munkhouse、Jeanne Boyarski、Bert Bates和Max Habbibi。

感谢Valentin Crettaz，不仅因为他是一位出色的技术编辑和代码贡献者（正如上面所说的），还因为是他最早把jQuery介绍给我。

跟往常一样，还要向我的妻子Jay、狗狗Little Bear和Cozmo（它的照片使得本书更加生动）致以衷心的感谢。他们忍受了我像个影子一样存在，在写书的几个月时间里，我们虽然生活在一起，但我却很少把目光从MacBook Pro的键盘上移开，抬头看上他们一眼。

最后，感谢我的合作者Yehuda Katz，没有他就不可能有这个项目，同时也感谢John Resig和其他jQuery和jQuery UI的贡献者们。

Yehuda Katz

首先，感谢我的合著者Bear Bibeault，他丰富的写作经验让我受益匪浅。他拥有天才的写作技巧，解决专业出版物中疑难问题的能力令人印象深刻，这是本书得以完成的极大保障。

说到本书的完成，我必须感谢我亲爱的妻子Leah，我花了这么长的时间写书而没有陪伴她，让她孤独地忍受了漫漫长夜和每个周末，对此我感到很愧疚。本书得以完稿离不开她的付出，而且，是她的支持使我走过了本项目最艰难的阶段。我爱你，Leah。

其次，没有jQuery库，就没有本书。感谢jQuery的创建者John Resig，他改变了客户端开发的面貌，并减轻了全球Web开发者的工作量（不管相信与否，在中国、日本、法国等许多国家，我们都拥有相当大的用户群）。我把他当成好朋友，在我完成这个艰巨任务的过程中，这位才华横溢的作者对我帮助甚多。

jQuery的面世离不开令人钦佩的社区用户和核心团队成员，包括开发组的Brandon Aaron和Jörn Zaefferer，推广组的Rey Bango和Karl Swedberg，负责jQuery UI（用户界面）的Paul Bakaus，和我一起在插件组工作的Klaus Hartl和Mike Alsup。这个强大的程序员团队把jQuery框架从紧凑、简单的核心操作基础库，推进为世界级的JavaScript库，几乎任何需求都有着由用户贡献的（模块化的）支持。jQuery的贡献者太多了，以至于我可能会漏掉很多人的名字。可以肯定，如果没有围绕这个库而兴起的独一无二的社区，我就不会写这本书，真不知道该如何感谢你们。

最后，我想感谢我的家人，迁居到西海岸以来，我没能经常回去看望家人。兄弟姐妹间的手足情谊让我度过了快乐的成长时光，家人的信任使我坚信能够克服任何困难。妈妈、Nikki、Abie，还有Yaakov：谢谢你们，我爱你们。

关于本书

更少的代码，更多的功能。

一言以蔽之，本书的目的是：帮助你学习如何在网页中以更少的脚本来实现更多的功能。本书的两位作者，一位是 jQuery 的狂热用户，另一位是 jQuery 贡献者和布道者，都坚信 jQuery 是目前最好的 JavaScript 库，可以帮助你实现上述目标。

本书致力于让你在快速高效地运行 jQuery 的同时，体验到写代码的乐趣。本书讨论了核心的 jQuery 库和配套的 jQuery UI 库的所有 API，每个 API 方法都以容易理解的语法块呈现，并描述了方法的参数和返回值。其中包含了很多高效使用 API 的例子，而且对于重要概念，我们提供了称为“实验室”的页面来详细描述。通过这些全面而有趣的网页，你可以在实践中观察 jQuery 方法的玄妙之处，而无需手工编写代码。

所有的示例代码和实验室页面都可以从这里下载：http://www.manning.com/jQueryinActionSecondEdition 或 http://www.manning.com/bibeault2。

我们可以用一堆营销的行话来告诉你本书有多么棒，但是你并不想浪费时间来听这些，对吧？你想要的是获取更多 jQuery 的知识，好吧，这正是本书的意图。

闲话少叙，请你继续阅读吧！

读者对象

本书面向各层次的 Web 开发人员，帮助他们使用 JavaScript 创建很酷的交互性 Web 应用程序，而不需要从头写那些原始的、依赖于浏览器特性的客户端代码。

所有希望借助 jQuery 的强大功能，创建高交互性和可用性的 Web 应用程序，以取悦客户的开发人员，都可以从中受益。

可能 Web 开发新手会发现一些章节有点深奥，但并不妨碍他们深入学习。我们在附录中介绍了 JavaScript 的基本概念，这将有助于最大限度地发挥 jQuery 的潜能。Web 开发新手只需理解一些关键概念，就会发现 jQuery 是很容易学的。对高级开发人员而言，jQuery 则是一个强大的武器。

无论 Web 开发的新手还是老将，客户端的程序员掌握 jQuery 都会受益匪浅。我们确信，本书的内容将有助于你快速掌握这门知识。

章节概述

本书的组织形式立足于帮助你以最高效的方式学习 jQuery 和 jQuery UI 的知识。它从 jQuery

创建时所采用的设计理念入手，快速过渡到 jQuery API 的基本概念。然后讲授 jQuery 各方面的知识，从事件处理到 Ajax 请求，这将帮助你书写规范的客户端代码。除此之外，本书还探讨了配套的 jQuery UI 库。

本书分为两部分：第一部分涵盖了核心的 jQuery 库，第二部分介绍 jQuery UI 库。第一部分包含 8 章。

第 1 章我们将了解 jQuery 的理念，以及它是怎么遵循现代编码原则的，比如不唐突的 JavaScript。我们讨论了采用 jQuery 的理由，概述了它的工作原理。我们深入关键概念，比如文档就绪处理程序、实用函数、DOM 元素的创建，以及如何扩展 jQuery。

第 2 章介绍 jQuery 包装集（wrapped set），这是 jQuery 运行的核心概念。我们将学习如何通过丰富而强大的 jQuery 选择器，从页面上选取元素并创建这些包装集（可被操作的 DOM 元素集合）。这些选择器使用标准的 CSS 选择符号，这使得它们非常强大，而这些 CSS 知识我们可能早已熟知。

第 3 章我们将学习如何使用 jQuery 包装集来操作页面 DOM 元素。这将包含改变元素的样式和特性、设置元素内容、移动元素、从头创建元素以及处理表单元素。

第 4 章展示了如何使用 jQuery 极大地简化页面上的事件处理。毕竟，响应用户事件使交互式 Web 应用变得可能，那些跨浏览器实现复杂事件处理的开发人员，一定会感激 jQuery 为这一领域带来的便利。本章还详细研究了高级事件处理概念，比如事件命名空间、自定义事件的触发和处理，甚至创建“live”事件处理程序，并组成一个完整的项目示例。

第 5 章的主题是动画和特效。我们将看到 jQuery 使创建动画特效不仅没有痛苦，而且高效且充满乐趣。为顺序执行动画准备的函数队列，以及普通的函数回调也在讨论之列。

第 6 章将学习 jQuery 提供的实用函数和标志，这些内容不仅适合页面开发人员，也适合为 jQuery 编写扩展和插件的开发人员。

第 7 章关注扩展和插件的开发。我们会看到 jQuery 使其变得极其简单，无需深奥的 JavaScript 或 jQuery 知识，任何开发人员都可以编写扩展，也会看到为什么需要将可重用的代码封装为 jQuery 扩展。

第 8 章关注现代 Web 应用开发中一个非常重要的领域：发起 Ajax 请求。我们将看到 jQuery 使网页上的 Ajax 开发变得如此简单，并防止我们掉入常见的陷阱，同时大大简化了使用常用数据类型的 Ajax 交互（比如返回 JSON 结构）。另一个综合示例，则涵盖了我们所学的 jQuery 的各个方面。

第二部分包含 3 章，我们将学习 jQuery 配套的界面库 jQuery UI。

第 9 章介绍了 jQuery UI 库，解释了如何配置和获取该库的代码定制版本，以及如何使用可视主题样式来实现不同的界面效果。我们剖析了可视主题，不仅可以了解它们是如何工作的，还可以修改主题以满足我们的需求。围绕本章我们讨论了 jQuery UI 库为 jQuery 核心所做的扩展，以及如何利用这些扩展的优势来增强核心方法。

第 10 章研究了 jQuery UI 提供的鼠标交互能力。无论是从元素拖放，还是元素排序、选择以及改变大小等操作，它都提供了支持。

最后，第 11 章全面阐释了 jQuery UI 控件集，这个部件集扩展了现有的页面输入机制。包括从类似按钮的简单控件到复杂控件，比如日期选择器、自动完成控件、选项卡面板以及对话框控件等。

我们在附录中介绍了 JavaScript 的基本概念，比如函数上下文和闭包。对于新手或者希望回顾旧知识的开发人员，理解这些概念将有助于更高效地使用 jQuery。

页边图标

在本书中，为了说明 jQuery 和 jQuery UI 的概念，我们编写了独特的实验室页面。这些实验室页面是可交互的网页，附有可下载的示例代码，你可以在本机打开并运行它们。



引入新的实验室页面时，页面左侧边栏处会有一个实验室图标（左侧所示锥形瓶图标），便于你能够在书中一眼看出实验室页面首次引入的位置。在下载的示例代码首页包含了实验室页面的链接。

你还可以通过远程站点 http://www.bibeault.org/jqia2 或图灵社区 www.ituring.com.cn 访问实验室页面以及示例代码的其余部分。



另外一个贯穿全书的图标是练习图标（三角形和铅笔），它们指出你需要做练习的地方。通常情况下，这些练习与特定的实验室页面相关联，但有时它们是本书中其他代码的逻辑扩展，或者是独立的简单练习题。完成这些练习有助于你更加深入地了解 jQuery。

代码约定

所有代码清单或者正文中的源代码都采用这样的字体（`jQuery Code`），以示区分。正文中的方法和函数名、对象属性、XML 元素以及节点属性也以这种字体呈现。

有些例子中，原始的源代码已经被排版以适合在页面上显示。一般来说，源代码样式的改变主要是由于页面宽度的限制，但有时你会发现书中的代码和下载的源代码格式略有差异。在少数情况下，为了不改变长代码行的含义，不会对它们进行重排，而是采用续行记号。

代码清单中的注释是为了强调重要的概念。在很多情况下，代码中带圈的数字与随后正文中出现相同带圈数字的地方一一对应。

代码下载

本书中所有的源代码示例（包括那些没有在正文中出现的额外示例），都可以从 http://www.manning.com/jQueryinActionSecondEdition 下载。为了方便那些无法在本地运行示例代码的开发人员，我们为示例代码创建了一个在线版本 http://www.bibeault.org/jqia2/。

本书中的代码示例被组织成 Web 应用程序，每章独立一块，方便使用者随时通过本地 Web 服务器获取服务，比如 Apache 的 HTTP 服务器。你可以将下载的代码解压缩到一个文件夹，并设置此文件夹为应用程序根目录。启动页面就是这个目录下的 index.html 文件。

除了描述 Ajax 请求的第 8 章以及 jQuery UI 的少数几章，大部分的示例都可以在浏览器直接打开，而无需使用 Web 服务器。Ajax 示例需要使用 Apache 所不具备的后端交互能力，因此要想在本地运行这些示例，需要为 Apache 安装 PHP 服务，或者使用可以执行 Java 小程序（Java servlets）或 JSP（JavaServer Pages）的服务器，比如 Tomecat。文件 chapter8/tomcat.pdf 将指导你轻松地安装 Tomcat 服务器以便运行 Ajax 示例。

所有的代码示例在各个主流浏览器下已测试通过，包括 IE 7/8、Firefox 3、Safari 3/4 以及 Google Chrome。

交流反馈

购买本书后，你可以访问 http://www.manning.com/jQueryinActionSecondEdition 发表对本书的评论，询问技术问题以及获取其他用户的帮助。[①]

① 读者也可以访问图灵社区（ituring.com.cn）发表对本书的评论。——编者注

关于封面插图

本书封面上的画像名称是“守望者”，摘自200百年前法国人J.G.St.Saveur所著的《旅行百科》。在那个年代，人们对为了获得乐趣而旅行感到很新鲜，类似的旅行指南大受欢迎，旅行者和足不出户的人们从旅行指南上了解世界各地的风土人情，其中还包括法国士兵、公务员、手工艺者、商人和农民在不同地区的服装样式和制服样式。

《旅行百科》通过各种各样的图片生动地描绘了200年前世界其他城镇的不同特色。人们彼此孤立，拥有各自的方言和语言。可以很容易地通过人们的语言和穿着，判定其居住的地方是城镇还是乡村，从事的职业以及拥有的地位。

从那时起，服饰样式有了很大的变化，各个地域原有的多种多样的服饰正在逐渐消失。现在，我们通常很难区分两个不同地区的人。也许，乐观地看，我们是用文化和外表上的多样性来换取更加多元化的生活，或者说多元化的、充满趣味的智慧人生。

这本两个世纪前的旅行指南充分展示了各地多姿多彩的地域风情，Manning出版社用这本指南中的一幅插图作为封面，以赞美计算机行业的创新性、开拓精神和蕴含的无穷乐趣。

目　　录

Part 1

第一部分

核心 jQuery

当人们听说 jQuery 这个名称时，就会自然而然地想到核心 jQuery 库；然而 jQuery 的意义不仅于此，它创建了一个围绕核心 jQuery 的生态系统，比如衍生的 jQuery UI（本书第二部分讨论的主题）、官方插件（参见 http://plugins.jquery.com/）以及大量的非官方插件，我们可以通过搜索引擎很容易地找到它们（在 Google 中搜索“jQuery plugin”会有超过 400 万项结果！）。

就像苹果的 iPod 一样，如果它本身不够优秀，那么为它扩展的第三方产品市场也就不会存在一样，核心 jQuery 库是这一切繁华的基础。

本书第一部分共 8 章，我们将会从头学习核心 jQuery 库。当你完成这几章的学习后，你会发现 jQuery 库一应俱全，这个客户端工具极其强大可以随时应用到任何 Web 开发项目中。你可能还会使用 jQuery 插件库（就像 iPod 配件一样），它们一定有自身存在的价值。

因此请翻开新的一页，仔细阅读并准备好为你的 Web 开发注入活力，你会发现这不仅简单而且充满乐趣！

第 1 章

jQuery 基础

本章内容

- ❑ 为什么使用 jQuery
- ❑ 不唐突的 JavaScript 的含义
- ❑ jQuery 的基本原理和概念
- ❑ 结合其他 JavaScript 库使用 jQuery

JavaScript 从诞生那天起就被很多 Web 开发人员嘲笑为“玩具”语言，然而在过去的几年时间里，人们逐渐对高交互性的下一代 Web 应用（也被称为关注富互联网应用或者基于 DOM 的脚本应用）以及 Ajax 技术产生了热情，从而使 JavaScript 又重获关注。由于客户端开发人员已经抛弃剪切和粘贴 JavaScript 的开发方式，转而使用方便快捷、功能完善的 JavaScript 库，因此这门语言也得到了长足的快速发展。这些库彻底解决了跨浏览器问题，并提供了新颖的、改进了的 Web 开发模式。

作为一个相对年轻的 JavaScript 库，jQuery 以迅雷不及掩耳之势席卷了整个 Web 开发社区，很快赢得了几家大公司的支持并被应用于其关键的产品中。jQuery 的一些典型用户包括 IBM、Netflix、Amazon、Dell、Best Buy、Twitter 美国银行以及很多其他的知名企业。微软甚至把 jQuery 和 Visual Studio 工具一块发布，而诺基亚则在所有的手机平台以及 WRT（Web Runtime Component）上使用 jQuery。

这些都是 jQuery 足以炫耀的资本！

和其他更加关注智能化的 JavaScript 工具包不同，jQuery 的目标是改变 Web 开发人员创建高交互性页面的方式。设计者无需花费时间纠缠 JavaScript 复杂的高级特性，而是使用现有的知识比如 CSS（Cascading Style Sheets，层叠样式表）、HTML（Hypertext Markup Language，超文本标记语言)、XHTML（Extensible Hypertext Markup Language，扩展的超文本标记语言）以及原生的 JavaScript 直接操作页面元素，从而实现快速开发。

本书中，我们会深入学习 jQuery 为高交互性 Web 应用程序开发者提供的功能。下面来探索 jQuery 为 Web 开发盛宴带来的实实在在的好处吧。

你可以从 jQuery 网站（http://jquery.com/）获取最新版本的 jQuery。安装 jQuery 相当简单，只需将它放在应用程序目录并在页面中使用 HTML 标签`<script>`来包含它，就像下面这样：

```
<script type="text/javascript"
        src="scripts/jquery-1.4.js"></script>
```

本书中使用的 jQuery 版本可以从下载的示例代码中找到（http://www.manning.com/bibeault2）。[①]

1.1　用少量代码实现丰富的功能

如果你曾经尝试为页面添加一些动态的功能，那么你就会发现实现的模式经常是相同的，首先选择一个元素（或者一组元素），然后以某种方式操作这些元素。你可以显示或隐藏元素，给元素添加 CSS 类，让元素动起来或者改变元素的特性。

使用原始的 JavaScript 完成这些任务中的任何一个，都需要数十行的代码，jQuery 的开发者们专门为此创建了 jQuery 库来简化这些常见任务。例如，有过处理单选按钮分组经验的开发人员都知道，找出分组中哪个元素被选中并获取它的 value 特性是非常乏味的；首先需要找出单选按钮分组，其次获取由这些单选按钮元素组成的数组并逐个遍历，找出哪个元素拥有 checked 特性，最后获取此元素的 value 特性。

过去人们通过如下代码来实现：

```
var checkedValue;
var elements = document.getElementsByTagName('input');
for (var n = 0; n < elements.length; n++) {
  if (elements[n].type == 'radio' &&
      elements[n].name == 'someRadioGroup' &&
      elements[n].checked) {
    checkedValue = elements[n].value;
  }
}
```

而 jQuery 只需要一行代码就能实现相同的功能：

```
var checkedValue = $('[name="someRadioGroup"]:checked').val();
```

如果这段代码看起来有点神秘请不用担心。很快你就能搞懂它的工作原理，并创建出简洁而强大的 jQuery 代码来武装你的页面。下面简单看一下这个强大的代码片段是如何工作的。

首先，找出所有 name 特性值为 someRadioGroup 的元素（别忘记单选按钮分组是由一系列拥有相同名称的单选按钮元素组成的），然后过滤这个集合从而得到处于 checked 状态的那个单选按钮，并获取它的 value 特性。（只可能有一个符合条件的元素，因为浏览器只允许选中同一分组中的一项单选按钮。）

这条 jQuery 语句的真正威力来自选择器，它是一个识别页面元素的表达式。它帮助我们轻松找到并获取需要的元素。在上面的例子中，就是单选按钮分组中被选中的元素。

如果你还没有下载示例代码，现在就去下载吧。下载链接就在本书的官方网站：http://www.manning.com/bibeault2（别忘记链接中最后的字符 2）。把下载的代码解压缩到任意文件夹之后，可以打开单独的文件，也可以通过根目录下的索引页（index.html）访问任意文件。

在浏览器中打开文件 chapter1/radio.group.html，如图 1-1 所示，这个示例展示了如何使用 jQuery

① 读者也可以登录图灵社区下载源代码，网址：www.ituring.com.cn。——编者注

获取单选按钮分组中被选中的元素。

这个简单的例子应该可以使你信服，jQuery 能够以轻松的方式创建高度交互的下一代 Web 应用程序。随着对本书各章的学习，我们将会逐渐揭开 jQuery 的神秘面纱，让你全面感受 jQuery 驾驭页面的强大能力。

我们很快就会学习如何创建 jQuery 选择器，正是它使得本例如此简单，但是先来了解一下 jQuery 作者对于如何在页面上最有效地使用 JavaScript 有什么样的想法。

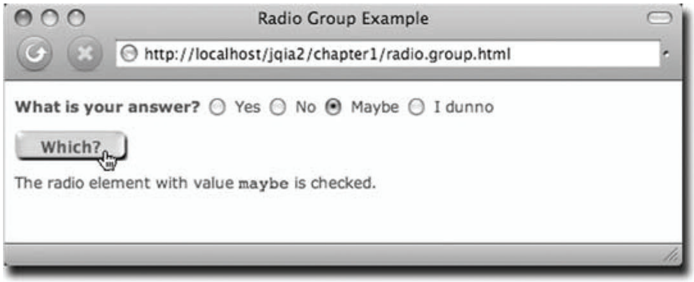


图 1-1　使用一条 jQuery 表达式轻松获取被选中的单选按钮

1.2　不唐突的 JavaScript

记得在 CSS 出现之前糟糕的往日吗？那时我们不得不在 HTML 页面中混合使用样式标记和文档结构标记。任何一个网页开发人员，无论从事开发的时间长短，都一定会对这种陈旧的开发方式深恶痛绝。

CSS 的出现让 Web 开发人员可以将样式信息从文档结构中分离出来，而且摒弃了老式的 `<font>`等标签[①]。将样式从文档中分离出来不仅有利于页面的维护，而且能够使我们只需通过切换样式表就能完成页面的换肤功能。

很少有人愿意再将样式嵌入 HTML 元素中，然而下面的代码却依然很常见。

```
<button
  type="button"
  onclick="document.getElementById('xyz').style.color='red';">
    Click Me
</button>
```

可以清楚地看到，这个按钮的样式（包括按钮标题的字体）不是通过废弃的`<font>`标签或者其他样式标签设置的，而是通过某种应用到页面上的 CSS 规则（没有列出）设置的。尽管这里的声明没有把样式和结构混合在一起，但是它把单击按钮时需要执行的 JavaScript 脚本（本例中，脚本把 `id` 值为 `xyz` 的 DOM 元素的颜色改为红色）放在了按钮标签的 `onclick` 特性中，这使得行为和结构混合在了一起。

下面来看看怎么改进这种情况。

① HTML 4.01 不赞成使用`<font>`标签，而 HTML 5 已经不支持这个标签了。（以下注释如非特别指明，均为译者注。）

1.2.1 行为和结构分离

就像在 HTML 文档中应该把样式从结构中分离出来一样，我们有同样的理由把行为从结构中分离出来。

理想情况下，HTML 页面应该按照图 1-2 的方式来组织结构、样式和行为 3 部分，每一部分都有属于自己的位置。

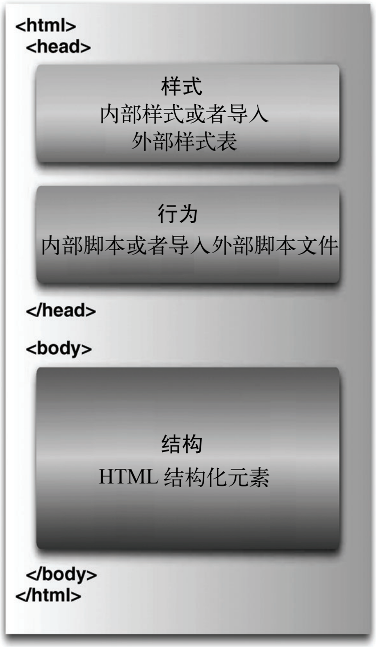


图 1-2 将结构、样式和行为整齐地放置在页面的不同部分，最大限度地确保了可读性和可维护性

这种策略被称为不唐突的 JavaScript，是由 jQuery 的开发者引入公众视野的，已经扎根于所有主要的 JavaScript 类库中，它能帮助页面开发者在页面中实现这种有效的分离。随着 jQuery 对这一概念的推广，jQuery 的核心也在逐步优化以便更加容易地生成不唐突的 JavaScript 代码。不唐突的 JavaScript 认为，任何存在于 HTML 页面`<body>`内部的 JavaScript 代码都是错误的，这些错误代码或者存在于 HTML 元素特性（比如 onclick 特性）中，或者存在于页面`<body>`脚本块中。

你或许会问："如果不使用 `onclick` 特性，如何为按钮添加行为呢？"考虑下面按钮元素的变化：

```
<button type="button" id="testButton">Click Me</button>
```

这就简单多了！但你会发现，现在的按钮不能做任何事情了。你可以一直不停地单击它，但是没有任何反应。

下面就来解决这个问题。

1.2.2 分离脚本

我们可以将按钮的行为移到<head>标签里的脚本块中，而不是把它们写在标记里，这样脚本就被移出了<body>之外，如下所示：

```
<script type="text/javascript">
  window.onload = function() {
    document.getElementById('testButton').onclick = function() {
      document.getElementById('xyz').style.color = 'red';
    };
  };
</script>
```

我们把脚本放在页面的 onload 处理函数中，并将一个内联函数赋给按钮元素的 onclick 特性。

之所以将这段脚本放在 onload 处理函数中（而不是直接放在内联代码块中），是因为我们要确保在使用按钮元素之前，它已经存在于页面中了。（在 1.3.3 节中，我们将看到 jQuery 提供了更好的地方来放置这些代码。）

注意 出于对性能的考虑，也可以把脚本块放在文档主体的底部，不过现代浏览器的速度都已经很快了，这么做的性能差异可以忽略不计。这里重点强调的概念是避免在结构性的元素中嵌入描述行为的代码。

如果本例的代码看起来不太好理解（比如函数字面值与内联函数的概念），也不要担心！为了帮助你更高效地使用 jQuery，本书附录包含了你需要掌握的重要 JavaScript 概念。本章的剩余部分将探讨 jQuery 如何帮助我们更加轻松和快速地编写上述代码，并保持代码的通用性。

不唐突的 JavaScript 是一项强大的编程技术，为 Web 应用开发引入了清晰的职责划分，但这样做也是有代价的。你可能已经注意到，与直接在按钮标记中放置脚本相比，不唐突的 JavaScript 需要编写更多行代码来实现同样的功能。不唐突的 JavaScript 可能会增加代码的行数，并且它还需要客户端脚本遵守一些规则并且应用良好的编码模式。

但这未尝不是一件好事，它能够促使我们以编写服务器端代码的严谨态度来编写客户端代码。但那需要做很多额外的工作——如果没有 jQuery 的话。

如前所述，jQuery 团队尤其注重以简单和愉快的方式在页面上使用不唐突的 JavaScript，而无需付出大量精力编写大段代码。我们将发现，有效利用 jQuery 能够有助于用更少代码实现更多的功能。

事不宜迟，下面就开始介绍如何在 jQuery 的帮助下轻松地为页面添加丰富的功能。

1.3 jQuery 基础

jQuery 主要关注从 HTML 页面上获取元素并对它们进行操作，这是 jQuery 的核心。如果你熟悉 CSS，那么对选择器的强大功能一定印象深刻，它通过类型、特性或在文档中的位置来描

述元素集合。有了 jQuery，你就能够利用现有知识发挥选择器的强大威力，极大地简化 JavaScript 代码。

jQuery 的首要目标就是确保在所有主要浏览器下都能流畅地运行代码。很多 JavaScript 难题（比如等待页面加载完毕后才能执行页面操作）都已经被 jQuery 轻松地解决了。

如果需要更多的功能，jQuery 还通过插件提供了简单而强大的内置方法，以扩展其现有的功能。很多 jQuery 新手发现他们在学习的第一天就能自己编写插件了。

下面开始学习如何借助 CSS 的现有知识来生成强大而简洁的代码。

1.3.1　jQuery包装器

当为了使设计和内容分离而把 CSS 引入 Web 技术的时候，需要以某种方法从外部样式表中引用页面中的元素组。这种方法就是选择器，它可以根据元素的类型、特性或者元素在 HTML 文档中的位置来精确地描述元素。

熟悉 XML 的开发者可能会联想到使用 XPath 选择 XML 文档中的元素。CSS 选择器使用了同样强大的概念，但为了适用于 HTML 页面而做了优化，从而更加简洁且更易理解。

比如，选择器

```
p a
```

引用位于<p>元素内的所有超链接（<a>元素）集合。jQuery 使用相同的选择器，不仅支持常用的 CSS 选择器，而且支持一些没有被所有浏览器完全实现的选择器，甚至包括 CSS3 中定义的一些强大的选择器。

要选择一组元素，只需使用如下的简单语法将选择器传递给 jQuery 函数。

```
$(selector)
```

或者

```
jQuery(selector)
```

可能一开始你会觉得`$()`符号有点儿奇怪，不过大部分 jQuery 用户会很快喜欢上这种简洁的语法。比如，选择位于<p>元素内的一组链接，可以使用如下代码：

```
$("p a")
```

`$()`函数（`jQuery()`函数的别名）返回特殊的 JavaScript 对象，它包含与选择器相匹配的 DOM 元素数组，这个数组中的元素是按照在文档中的顺序排列的。这个对象拥有大量有用的预定义方法，可作用于已收集的元素集合。

在编程用语中，这种构造方法称为*包装器*（wrapper），因为它包装了收集到的元素并添加了扩展功能。我们使用术语 *jQuery 包装器*或者*包装集*，来指这些可以通过 jQuery 定义的方法操作的含有匹配元素的集合。

假如想隐藏所有类名为`notLongForThisWorld`的`<div>`元素，就可以使用以下 jQuery 代码：

```
$("div.notLongForThisWorld").hide();
```

这样的方法有很多，通常称为 *jQuery 包装器方法*，它们有一个特点——当执行完毕后（比

如隐藏操作）会返回相同的一组元素，以便为另一个操作做准备。例如，假设除了隐藏这些元素，我们还想为每一个元素增加一个新的类 removed。可以这样写：

```
$("div.notLongForThisWorld").hide().addClass("removed");
```

这些 jQuery 作用链可以无限延续。jQuery 作用链绵延数十种方法的示例也不罕见。因为每一种方法都作用在与最初的选择器相匹配的元素集上，所以没有必要遍历元素集。jQuery 已经为我们在后台完成了这一切！

尽管选择的对象组通过一个高度复杂的 JavaScript 对象来呈现，但是如果有必要的话，也可以把它当做普通的元素数组来对待。因此，下面两段代码的结果相同：

```
$("#someElement").html("I have added some text to an element");
```

和

```
$("#someElement")[0].innerHTML =
  "I have added some text to an element";
```

由于使用了 ID 选择器，因此选择器只能匹配一个元素。第一个示例使用 jQuery 的 html()方法，用一些 HTML 标记替换 DOM 元素的内容。第二个示例使用 jQuery 获取元素数组，通过数组下标 0 选择第一个元素，然后将值赋给一个常见的 JavaScript 属性 innerHTML 来替换内容。

如果想让匹配多个元素的选择器达到相同的效果，下面两个代码片段将会产生相同的结果（但第二个不是 jQuery 推荐的编程方式）。

```
$("div.fillMeIn")
  .html("I have added some text to a group of nodes");
```

和

```
var elements = $("div.fillMeIn");
for(var i=0;i<elements.length;i++)
  elements[i].innerHTML =
    "I have added some text to a group of nodes";
```

随着逻辑逐渐复杂，使用 jQuery 作用链可以不用改变预期结果而减少代码行数。此外，jQuery 不仅支持我们已经知道并且喜爱的选择器，而且还支持更高级的选择器（作为 CSS 规范定义的一部分），甚至是一些自定义选择器。

表 1-1 给出了一些示例。

表 1-1　选择器示例

选　择　器	结　　果
`$("p:even")`	选择所有偶数行的<p>元素
`$("tr:nth-child(1)")`	选择每个表格的第一行元素
`$("body > div")`	选择作为<body>直接子节点的<div>元素
`$("a[href$= 'pdf ']")`	选择链接到 PDF 文件的超链接元素
`$("body > div:has(a)")`	选择作为<body>直接子节点的、包含超链接（<a>）的<div>元素

这太强大了！

你可以利用已经掌握的 CSS 知识来快速入门并提高，然后学习 jQuery 支持的更高级选择器。我们会在第2章详细介绍 jQuery 选择器，你也可以在 jQuery 站点找到选择器的完整列表 http://docs.jquery.com/Selectors。

选择 DOM 元素来进行操作是 Web 开发中的常见需求，但有些需求和 DOM 元素毫无关系。下面粗略考察除了元素操作之处 jQuery 还提供了哪些功能。

1.3.2 实用函数

尽管包装将要操作的元素是 jQuery `$()` 函数最常见用法之一，但却并不是它的唯一职责。`$()` 函数的另一个职责是作为一些通用实用函数集的*命名空间前缀*。由于以选择器为参数来调用`$()` 生成的 jQuery 包装器赋予了页面开发者如此强大的力量，以至于大部分的页面开发者很少会用到其中的一些实用函数。实际上，直到第 6 章我们才会详细地学习大部分实用函数以便为编写 jQuery 插件做准备。不过下面的几节将会用到一些实用函数，因此先来简单了解一下。

这些函数的符号初看起来可能会有点奇怪。比如，用来去除字符串前后空格的实用函数，其调用形式如下：

```
var trimmed = $.trim(someString);
```

如果你觉得`$.`前缀有点儿怪异，就记住`$`是 JavaScript 中的标识符，和其他的标识符没有什么不同。使用标识符 `jQuery`（而不用别名`$`）来调用这个函数，可能看起来就没那么奇怪了如：

```
var trimmed = jQuery.trim(someString);
```

现在可以清楚地看到，`trim()`仅仅是 `jQuery`（或它的别名`$`）命名空间下的一个函数。

注意 尽管这些元素在 jQuery 文档中被称为实用函数，但它们实际上是`$()`函数的方法[①]（是的，在 JavaScript 中函数也可以有自己的方法）。为了不和 jQuery 在线文档产生术语上的冲突，我们撇开这些技术差异，使用实用函数来描述这些方法。

我们会在 1.3.5 节中学习一个用来帮助扩展 jQuery 的实用函数，然后在 1.3.6 节中学习另一个可以让 jQuery 和其他客户端库共存的实用函数。但首先要看一下 jQuery 的`$()`函数的另一个重要职责。

1.3.3 文档就绪处理程序

当遵循不唐突的 JavaScript 原则时，行为就从结构中分离出来了，因此我们要在文档标记外部执行对页面元素的操作。为了达到这一目的，我们需要一种在页面 DOM 元素完全渲染后再执行操作的方法。在“单选按钮分组”示例中，必须在整个页面主体加载完毕后再应用行为。

传统上，`window` 实例的 `onload` 处理程序就是为了实现上述目的，该程序在整个页面加载

① 可以假设 jQuery 的内部实现如下：`function $(){}  $.trim=function(){}`。

完毕后执行代码。其典型的语法如下：

```
window.onload = function() {
  // 在这里执行操作
};
```

这样就能在文档完全加载后执行定义的代码。然而，浏览器会延迟执行 onload 的代码，不仅是在创建 DOM 树之后，而且在所有外部资源全部加载完毕，并且整个页面在浏览器窗口中显示完毕之后。这些资源不仅包括图片资源，而且包括嵌入在页面上的 QuickTime 和 Flash 视频资源，如今这类资源会越来越多。结果是，从首次浏览页面到 onload 脚本执行完毕，访问者将会经历一段严重的延迟。

更糟糕的是，如果图片或者其他资源需要很长时间来加载，则访问者就需要等待图片加载完成后才能使用丰富的页面行为。这会使不唐突的 JavaScript 的提议，在很多实际案例中遭遇滑铁卢。

更好的方式是，在执行脚本以便应用丰富行为之前，只需要等待文档结构被完全解析并且浏览器已经把 HTML 转化为 DOM 树。跨浏览器完成这个任务略有困难，但是 jQuery 提供了一个简单的方式：在 DOM 树（而不是外部资源）加载完毕之后立即触发脚本的执行。正式的语法结构（使用前面的隐藏元素的例子）如下所示：

```
jQuery(document).ready(function() {
  $("div.notLongForThisWorld").hide();
});
```

首先，使用 jQuery()函数包装 document 对象，然后调用 ready()方法并传递一个在文档就绪时执行的函数。

上述语法称为*正式语法*是有原因的，因为还有一个更常用的简写形式，如下所示：

```
jQuery(function() {
  $("div.notLongForThisWorld").hide();
});
```

通过传递一个函数给 jQuery()或者$()，我们通知浏览器直到 DOM 被完全加载完毕（仅仅是 DOM）后再执行此段代码。更棒的是，可以在一个 HTML 文档中多次调用此函数，浏览器会按照它们在页面中声明的顺序依次调用这些函数。与此相反，window 的 onload 机制只允许指定一个函数。如果包含的任何第三方代码出于自身考虑使用 onload 机制（并非最佳方法），那么这个限制可能导致难以发现的问题。

以上就是$()函数的另一种用法，现在看看它还能做什么。

1.3.4　创建DOM元素

目前为止可以看到，jQuery 的开发者通过赋予$()函数（仅仅是 jQuery()的别名）更多的职责，来避免向 JavaScript 命名空间引入大量的全局名称。不过，$()函数还有一个职责需要探讨。

可以通过向$()函数传递一个包含 HTML 标记的字符串动态地创建 DOM 元素。例如，可以创建一个新的段落元素，如下所示：

```
$("<p>Hi there!</p>")
```

但是凭空创建一个 DOM 元素（或者具有层级的元素）用处不大。通常在按照这种方式创建有层级的元素之后，会接着应用 jQuery 的 DOM 操作函数。下面探讨代码清单 1-1 中的示例。

代码清单 1-1　动态创建 HTML 元素

```
<html>
  <head>
    <title>Follow me!</title>
    <link rel="stylesheet" type="text/css"
          href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $("<p>Hi there!</p>").insertAfter("#followMe");
      });
    </script>
  </head>

  <body>
    <p id="followMe">Follow me!</p>
  </body>
</html>
```

❶ 在文档就绪处理器中创建 HTML 元素

❷ 已经存在的元素，动态创建的元素会在此元素之后

这个示例在文档主体中创建了一个名为 `followMe` ❷的 HTML 段落。在<head>中的脚本元素里创建了一个就绪处理器❶，用来把一个新创建的段落插入到 DOM 树中那个已有元素的后面，语法如下：

```
$("<p>Hi there!</p>").insertAfter("#followMe");
```

结果如图 1-3 所示。



图 1-3　动态创建和插入元素通常需要多行代码，不过 jQuery 可以使用一行代码轻松搞定

我们会在第 3 章中详细地考察全套的 DOM 操作函数，jQuery 提供了许多操作 DOM 的方式，以创建任何想要的 HTML 结构。

在学习了 jQuery 的基本语法之后，下面将学习 jQuery 库的最强大的一个功能。

1.3.5　扩展jQuery

`jQuery` 包装器函数提供了大量有用的方法，可以在页面上反复使用。但是没有任何一个库

可以满足用户的所有需求。试图满足所有需求的库是不合理的，这可能会导致一大堆庞大的、臃肿的、包含很少可用功能的代码，这只会把事情搞砸。

由于认识到这个概念的重要性，所以 jQuery 开发者努力提取大部分页面开发者都需要的功能，只在核心库包含这些功能。jQuery 的开发者还意识到每个页面开发者都有个性化需求，因此将 jQuery 设计成可以很容易通过附加功能来扩展的库。

可以编写一些函数来填补任何空白，但是一旦习惯 jQuery 的代码风格，我们会发现使用传统的方式来扩展功能是超级乏味的。可以使用 jQuery 提供的强大功能对其进行扩展，特别是在元素选择领域。

下面看一个具体的例子：jQuery 没有任何预定义函数来禁用一组表单元素。如果在应用中大量使用表单，就会发现下面的代码非常便利：

```
$("form#myForm input.special").disable();
```

幸运的是，通过扩展在调用`$()`时返回的包装器，jQuery 从设计上保证了其方法集很容易就得到扩展。下面请看如何通过创建一个新的`disabled()`函数实现上述功能的基本用法：

```
$.fn.disable = function() {
  return this.each(function() {
    if (this.disabled != null) this.disabled = true;
  });
}
```

这里引入了很多新的语法，但不要过于担心。只要继续学习以下几章，你将会很熟悉这些语法并会频繁使用这些基本的惯用语法。

首先，`$.fn.disable`表示使用`disabled`方法扩展`$`包装器。在函数内部，`this`关键字指向将要操作的 DOM 元素集合。

然后，通过调用包装器上的`each()`方法遍历每一个匹配的元素。我们会在第 3 章详细探讨这种方法及其类似方法。在传入`each()`的迭代器函数内，`this`是对当前遍历的特定 DOM 元素的引用。不要被嵌套函数内`this`指向不同对象所迷惑。多写一些扩展函数后你就会发现，`this`指向当前函数内的不同对象是相当自然的事情。（附录中也有对 JavaScript 中`this`关键字概念的解释。）

检查每一个元素是否具有`disabled`特性，如果有，就把该特性设为`true`。返回`each()`方法的执行结果（即包装器），从而使新创建的`disabled()`方法支持链式操作（正如许多 jQuery 原生方法那样）。可以这样编写代码：

```
$("form#myForm input.special").disable().addClass("moreSpecial");
```

从页面代码的角度看，新方法`disabled()`仿佛内置在 jQuery 库一样！这门技术格外强大，以至于很多 jQuery 新手在使用 jQuery 库的初期就能创建一些小的扩展了。

此外，jQuery 的企业级用户已经为 jQuery 扩展了多套有用的功能（也被称为*插件*）。我们会在第 7 章中详细介绍这种扩展 jQuery 的方法。

测试存在性

你可能见过这个用来判断一个项目是否存在的常见用法：

```
if (item) {
  // item 存在的处理程序
}
else {
  // item 不存在的处理程序
}
```

这里的逻辑是，如果 item 不存在，则条件表达式将等于 false。

尽管这在大部分的情况下都能正常工作，但 jQuery 的开发者还是认为这过于宽松和不严谨，推荐在$.fn.disable 示例中使用更加明确的测试方法：

```
if (item != null) ...
```

这个表达式会正确地测试 null 或者 undefined 值。

访问 jQuery 在线文档即可查看 jQuery 团队推荐的各种代码风格的完整列表：http://docs.jquery.com/JQuery_Core_Style_Guidelines。

在深入学习如何使用 jQuery 为页面增加活力之前，你可能想知道能否同时使用 jQuery 和 Prototype，或者其他同样使用$缩略符的库。下一节会揭示这个问题的答案。

1.3.6 jQuery与其他库共存

尽管 jQuery 所提供的功能强大的工具集足以满足大部分的需求，但有时还会遇到一种页面需要引入多个 JavaScript 库的情况。这种情况之所以会产生，可能是因为我们把一个使用了其他库的应用转换为使用 jQuery 库，也可能是我们想在页面中同时使用 jQuery 和其他库。

jQuery 开发团队明确表示，他们关注的是如何满足 jQuery 社区用户的需求，而非排挤其他库，因此提供了允许 jQuery 和其他库共存的方法。

首先，他们遵循最佳实践原则，避免引入大量的标识符而污染全局命名空间。因为大量的标识符不仅可能会与其他库冲突，也可能会与我们想在页面上使用的名字冲突。标识符 jQuery 和其别名$是 jQuery 为全局命名空间引入的唯一变量。在 1.3.2 节曾提到的作为 jQuery 命名空间一部分而定义的实用函数，就是出于这方面考虑的最好示例。

尽管其他库定义一个名为 jQuery 的全局标识符的可能性不大，但是在下面这种特殊情况下，jQuery 的开发者出于方便性考虑使用$作为 jQuery 的别名，却会带来麻烦。其他 JavaScript 库（比如著名的 Prototype 库）出于自身考虑会使用$名称。并且$名称是 Prototype 库运行的关键，这就产生了严重的冲突。

深思熟虑的 jQuery 作者通过一个名为 noConflict()的实用函数来消除这种冲突。在引入会产生冲突的库后，调用

```
jQuery.noConflict();
```

就可以保留$标识符给其他库使用了。

我们会在第 7 章详细阐述引入此实用函数后代码的细微差别。

1.4　小结

在本章对 jQuery 的快速介绍中，我们学习了许多内容，为深入探索使用 jQuery 轻松、快速地开发下一代 Web 应用程序做好了准备。

jQuery 通常对任何需要执行操作（除了那些微不足道的 JavaScript 操作）的页面都非常有用，而且也高度关注如何帮助页面开发者在页面中使用不唐突的 JavaScript。采用 jQuery 的解决方案，把行为从结构中分离出来，正如 CSS 把样式从结构中分离出来一样，它能使我们更好地组织页面，增加代码的灵活性。

尽管事实上 jQuery 仅仅向 JavaScript 命名空间引入了两个新的命名（`jQuery` 函数和其别名`$`），它却通过使 `jQuery` 函数高度多功能化，以及根据传入的参数调整执行的操作来提供大量的实用功能。

`jQuery()`有如下用途：

- ❑ 通过包装器方法选择和包装将要操作的 DOM 元素；
- ❑ 为全局实用函数提供命名空间；
- ❑ 从 HTML 标记创建 DOM 元素；
- ❑ 创建在 DOM 就绪后执行的代码。

jQuery 在页面上表现出色，它不仅尽量减少了对全局 JavaScript 命名空间的占用，而且提供了官方方法，以尽可能减少当冲突依然存在时对命名空间的占用问题，也就是当页面上另一个库（比如 Protorype）也要求使用$名称的时候。jQuery 在用户友好方面做得不错吧！

接下来的几章，我们将探索 jQuery 为富因特网应用开发人员提供的全部功能。下面即将开始学习之旅，下一章我们学习使用 jQuery 选择器快速、简单地找出那些需要操作的元素。

第 2 章

选择要操作的元素

本章内容

- 使用选择器选择 jQuery 需要包装的元素
- 创建并放置新的 HTML 元素到 DOM 中
- 操作元素包装集

在上一章中，我们讨论了 jQuery 函数的多种用法，涵盖了从选择 DOM 元素到定义在 DOM 加载完毕后执行的函数等各种功能。

本章，我们将通过剖析`$()`函数的两个最强大和最常用的功能——使用选择器选择 DOM 元素并创建新的 DOM 元素，来介绍如何取得要操作的元素。

交互式 Web 应用的许多功能都是通过对组成页面的 DOM 元素进行操作来实现的。但在操作这些元素之前，我们首先需要找到并选择它们。下面开始探索之旅，学习使用 jQuery 选择目标元素的各种方法。

2.1 选择将被操作的元素

在使用 jQuery 的任何方法（常被称为 jQuery *包装器方法*）时，首先要做的是选择页面中要操作的那些元素。有时，想要选择的元素集是很容易描述的，比如“页面上的所有段落元素”。有时，需要对它们进行比较复杂的描述，比如“所有 CSS 类名为 `listElement`、包含一个链接的列表元素，并且是列表中的第一个元素”。

幸好 jQuery 所提供的*选择器*语法十分通用，可以让我们优雅而精确地描述所需选择的元素集。你可能已经知道很多这样的语法：jQuery 使用你已经熟悉和喜爱的 CSS 语法，并经过扩展提供了一些自定义语法，来帮助完成复杂而又常见的选择操作。

为了帮助你学习元素选择，我们在本书的示例代码中包含了 jQuery 选择器实验室页面（即 chapter2/lab.selectors.html）。它允许你输入一个 jQuery 选择器字符串，然后（实时地）观察哪些 DOM 元素被选中。实验室页面显示的结果如图 2-1 所示（如果面板排列不整齐，你可能需要调整浏览器窗口的大小）。

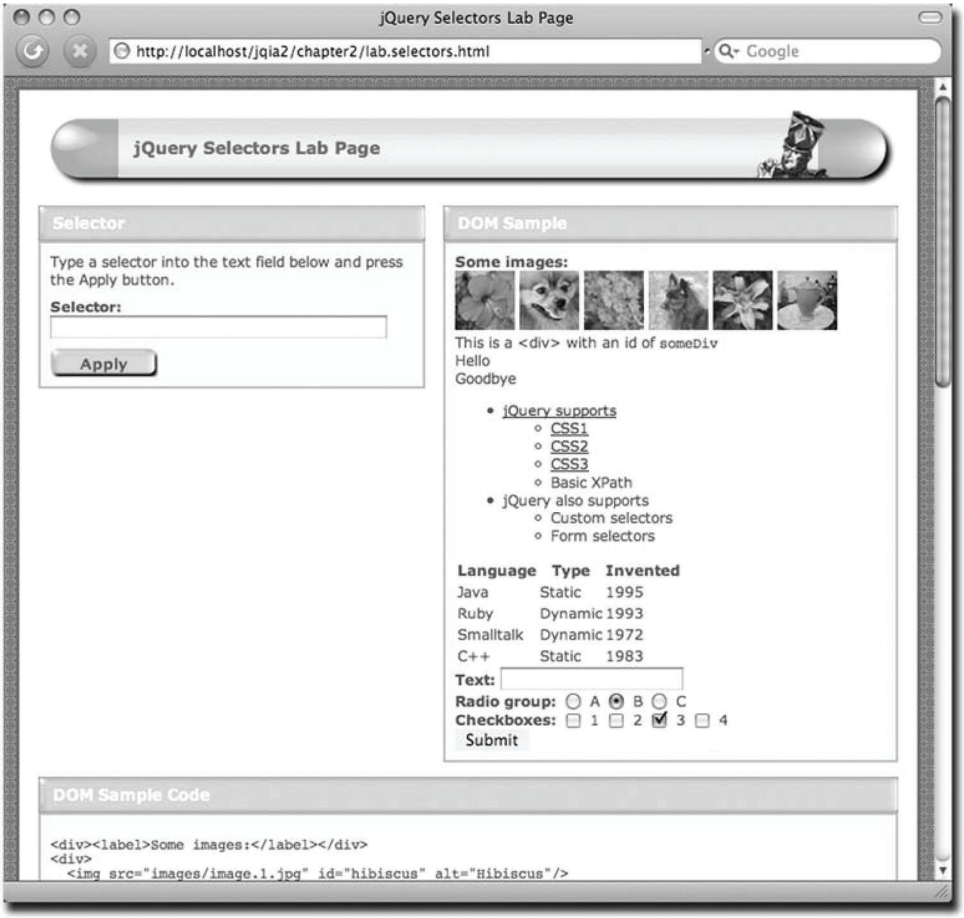


图 2-1　jQuery 选择器实验室页面允许实时观察任何指定选择器的行为

提示　如果你还没有下载示例代码，最好现在就去下载。如果你跟着实验室页面练习的话，会很容易消化和吸收本章的内容。请访问本书站点下载示例代码：http://www.manning.com/bibeault2。

图 2-1 中左上角的 Selector 面板包含一个文本输入框和一个按钮。在输入框中输入选择器字符串并单击 Apply 按钮，试验一下选择器实验室页面。例如，在文本框中输入字符串 `li`，然后单击 Apply 按钮。

输入的选择器（本例中是 `li`）将应用到页面右上角 DOM Sample 面板中的那部分 HTML 代码上。单击 Apply 按钮，实验室代码会立即执行，将所有匹配的元素添加类名 `wrappedElement`。在页面上定义的 CSS 规则确保所有应用此类的元素都高亮显示，边框变为红色，背景色变为粉

红色。单击 Apply 后，效果将与图 2-2 显示的一样，DOM 示例中所有`<li>`元素都高亮显示。

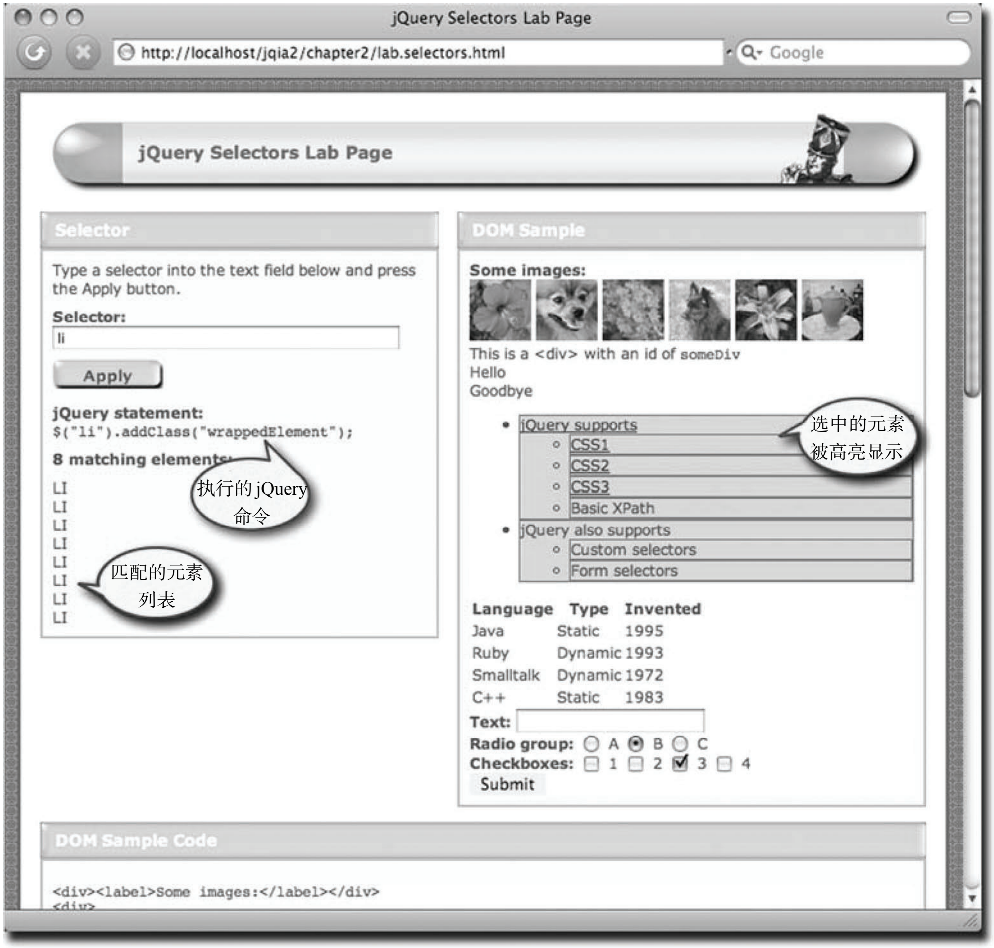


图 2-2　选择器 `li` 会匹配所有的`<li>`元素，应用后的显示结果如图所示

注意，示例中所有的`<li>`元素都已经高亮显示，执行的 jQuery 表达式以及匹配元素的标签名称都显示在了选择器输入框的下面。

用来生成 DOM 示例片段的 HTML 标记，显示在下面的 DOM Sample Code 的面板中。这可以帮助你验证针对 DOM 示例中的目标元素而写的选择器。

随着本章内容的推进，我们还会更详细地介绍此实验室页面。但是首先，作为本书的作者，必须承认我有意简化了一个重要的概念，下面会纠正这个问题。

2.1.1　控制上下文

目前为止，我们都假设传递给 jQuery 的`$()`函数的参数只有一个，但这仅仅是为了在开始的

时候尽量保持简单。事实上，这个函数在接受选择器或者 HTML 代码作为第一个参数时，也接受了第二个参数。如果第一个参数是选择器，那么第二个参数则指示该操作的上下文。

我们将在许多 jQuery 方法中看到，当一个可选的参数被忽略时，就会被一个默认的参数所替代。上下文参数 `context` 也是同样的道理。选择器作为第一个参数传递时（后面会讨论传递 HTML 代码的情况），默认把 DOM 树的所有元素作为该选择器的上下文。

通常这正是我们所需要的，因此这是一个不错的默认值。但是，有时我们只想搜索整个 DOM 树的一个子集。在这种情况下，需要指出 DOM 树的子集作为选择器应用的根节点。

选择器实验室页面为这种场景提供了一个很好的示例。你输入到文本框中的选择器，只会应用到 DOM Sample 面板加载的那些 DOM 树的子集上。

上下文参数可以是 DOM 元素的引用，也可以是包含 jQuery 选择器的字符串，或者是 DOM 元素包装集。（也就是说，我们可以把调用某个`$()`函数的结果传递给另一个`$()`函数——仔细想一想，这点儿逻辑应该不难理解。）

当选择器或包装集作为上下文参数时，选定的元素就成为了选择器（第一个参数）的上下文。我们可以通过标识多个这样的元素，来优雅地指定各种作为选择器上下文的 DOM 子树。

以实验室页面为例，假定选择器字符串保存在名为 `selector` 的变量中，应用选择器时，我们只想把上下文限制在示例的 DOM 树上，也就是 `id` 值为 `sampleDOM` 的`<div>`元素内。

如果我们像下面这样调用 jQuery 函数：

```
$(selector)
```

则选择器会应用到整个 DOM 树上，包含输入选择器的表单自身。这不是我们想要的结果。我们想把选择的范围限制在 DOM 树的子树内，该子树的根是 `id` 值为 `sampleDOM` 的`<div>`元素。因此要这样写：

```
$(selector,'div#sampleDOM')
```

这样就可以把选择器的应用范围限制在 DOM 树中指定的部分。

好了，我们已经知道了如何控制选择器应用的上下文范围，下面就从熟悉的领域——传统的 CSS 选择器开始，学习如何进行编码。

2.1.2　使用基本CSS选择器

由于要经常给页面元素应用样式，因此 Web 开发人员早已熟悉了几种选择表达式，这些表达式强大而实用，并且可以在所有浏览器下运行。这些表达式通过元素的 ID、CSS 类名、标签名以及页面中元素的 DOM 层级关系选择元素。

表 2-1 提供的一些例子可以帮你快速复习以前的知识。可以组合使用这些基本的选择器类型，来识别那些需要精确匹配的元素集。

表 2-1　一些简单的 CSS 选择器示例

示　例	描　述
`a`	匹配所有的链接元素（`<a>`）

（续）

示　　例	描　　述
#specialID	匹配 id 为 specialID 的元素
.specialClass	匹配拥有 CSS 类 specialClass 的元素
a#specialID.specialClass	匹配 id 为 specialID 并且拥有 CSS 类 specialClass 的链接元素
p a.specialClass	匹配拥有 CSS 类 specialClass 的链接元素并且这个元素是<p>的子节点

有了 jQuery，我们就可以使用驾轻就熟的 CSS 选择器来轻松地选择元素了。使用 jQuery 选择元素，只需将选择器传递给$()函数，就像下面这样：

```
$("p a.specialClass")
```

除了极少数例外，jQuery 完全兼容 CSS3 标准，因此按照这种方式选择元素更容易被大多数人所接受。遵循标准的浏览器通过样式表能够选中的元素，用 jQuery 选择器引擎也能选中。注意，jQuery 不依赖于它所在浏览器的 CSS 实现。即使浏览器没有正确地实现标准 CSS 选择器，jQuery 也会遵循 W3C 标准的规则来正确地选择元素。

jQuery 也允许我们使用逗号操作符将多个选择器合并成一个选择器。例如，要选择所有的<div>和所有<span>元素，可以这么做：

```
$('div,span')
```

现在可以做一些练习，打开选择器实验室页面，输入一些基本的 CSS 选择器试验一下，直到你觉得得心应手为止。

这些基本的选择器是强大的，但有时我们需要更加精确地控制需要匹配的元素。jQuery 勇于迎接挑战，提供了更高级的选择器。

2.1.3 使用子节点、容器和特性选择器

对于更高级的选择器，jQuery 也可以使用 Mozilla Firefox、IE7/8、Safari、Chrome 和其他现代浏览器都支持的最新 CSS 实现。这些高级选择器允许我们选择元素的直接子节点，在 DOM 中位于其他元素后面的元素，甚至其特性匹配一定条件的元素。

有时，我们只想选择某个元素的直接子节点。比如，只选择某个列表的直接列表元素，而不包含子列表中的列表元素。考虑以下出自选择器实验室页面中示例 DOM 的 HTML 代码：

```
<ul class="myList">
  <li><a href="http://jquery.com">jQuery supports</a>
    <ul>
      <li><a href="css1">CSS1</a></li>
      <li><a href="css2">CSS2</a></li>
      <li><a href="css3">CSS3</a></li>
      <li>Basic XPath</li>
    </ul>
  </li>
  <li>jQuery also supports
    <ul>
      <li>Custom selectors</li>
```

```
      <li>Form selectors</li>
    </ul>
  </li>
</ul>
```

假设我们想要选择远程 jQuery 站点的链接，而不是描述不同 CSS 规范的各种指向本地页面的链接。使用基本 CSS 选择器，我们会尝试这样写 `ul.myList li a`。然而，这个选择器会选取所有的链接，因为它们都是源自于同一个列表元素。

你可以在选择器实验室页面输入 `ul.myList li a`，单击 Apply 来对此进行验证。结果如图 2-3 所示。

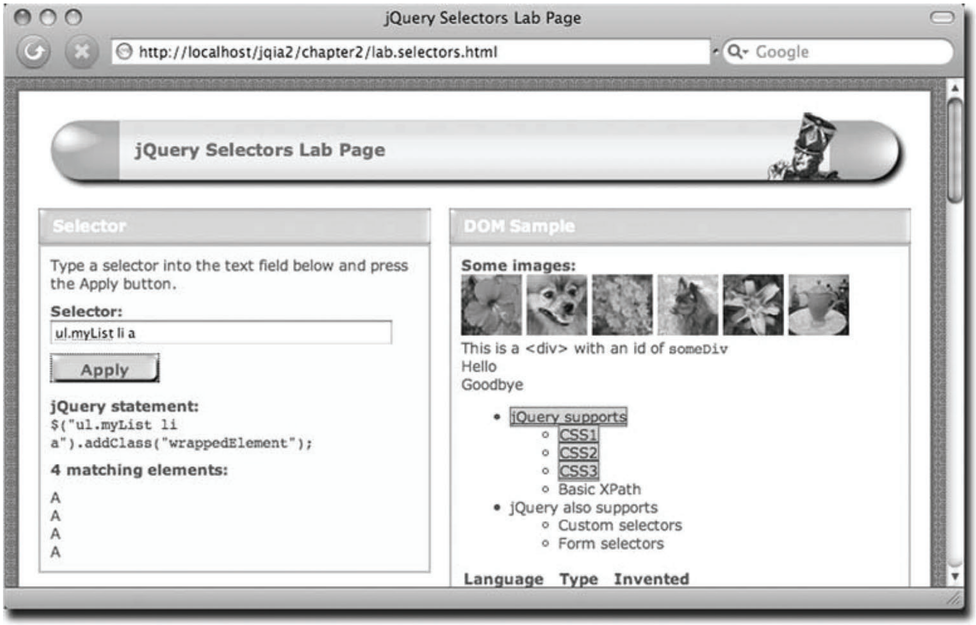


图 2-3　`ul.myList li a` 可以选中所有锚点标记，虽然它们所处的层次不同，但都是`<li>`元素的子节点

更高级的方法是使用子节点选择器，其中父节点和它的直接子节点通过右尖括号（>）隔开，如下所示：

```
p > a
```

此选择器只会匹配是<p>元素的直接子节点的链接。如果链接嵌入层次更深，比如位于<p>里面的<span>节点内，那就不会被选中。

回到示例上来，考虑如下选择器：

```
ul.myList > li > a
```

此选择器只会选择是列表元素直接子节点的链接，并且这个列表项是拥有 CSS 类 `myList` 的`<ul>`元素的直接子节点。排除包含在子列表中的链接，因为子列表项`<li>`的父节点`<ul>`元素并

不带有 CSS 类 myList，结果如图 2-4 所示。

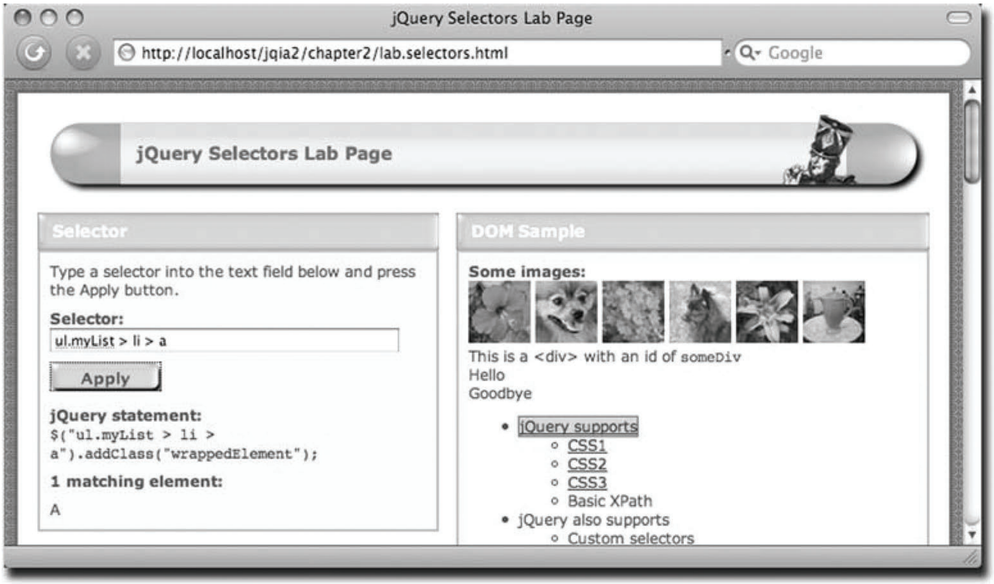


图 2-4　ul.myList > li > a 选择器只会匹配父节点的直接子节点

特性选择器也非常强大。假设我们想要为外部链接添加一个特殊的行为。再次看下之前探讨过的实验室页面示例的代码片段：

```
<li><a href="http://jquery.com">jQuery supports</a>
    <ul>
      <li><a href="css1">CSS1</a></li>

      <li><a href="css2">CSS2</a></li>
      <li><a href="css3">CSS3</a></li>
      <li>Basic XPath</li>
    </ul>
</li>
```

外部链接的 href 特性值都是以 http://开头的字符串，这使其与众不同。我们可以通过下面的选择器来匹配那些 href 特性值以 http:// 开头的链接：

```
a[href^='http://']
```

这会匹配 href 特性值以 http:// 开头的所有链接元素。插入符（^）用来指定匹配字符串开头的字符。这也是大部分正则表达式处理器使用的字符，表示匹配候选字符串开头的字符，记住这个用法应该很容易。

再次访问实验室页面（之前的 HTML 代码就摘自该实验室），在文本框中输入 a[href^='http://']，然后单击 Apply。注意只有一个 jQuery 的链接高亮显示。

还有其他使用特性选择器的方式。匹配某个元素是否拥有某个特性，而不论该特性的值是什么，我们可以这样使用：

```
form[method]
```

这会匹配拥有 method 特性的所有<form>元素。

匹配指定的特性值，使用如下语句：

```
input[type='text']
```

这个选择器会匹配 type 特性值为 text 的所有<input>元素。

我们已经看到如何“匹配特性值开头部分”的选择器。下面是另一个例子：

```
div[title^='my']
```

它选择 title 特性值以 my 开头的所有<div>元素。

对于“匹配特性值结尾部分”的选择器，该怎么做呢？下面就是：

```
a[href$='.pdf']
```

这是一个有用的选择器，用来定位所有指向 PDF 文件的链接。

还有一个选择器，可以查找特性值包含任意指定字符串的元素：

```
a[href*='jquery.com']
```

正如所料，这个选择器匹配指向 jQuery 站点的所有<a>元素。

表 2-2 显示了可以在 jQuery 中使用的基本 CSS 选择器。

表 2-2　jQuery 支持的基础 CSS 选择器

选　择　器	描　　述
*	匹配所有元素
E	匹配标签名为 E 的所有元素
E F	匹配标签名为 F 的所有元素，这些元素是 E 的子节点
E>F	匹配标签名为 F 的所有元素，这些元素是 E 的直接子节点
E+F	匹配标签名为 F 的所有元素，这些元素是 E 后面的第一个兄弟节点
E~F	匹配标签名为 F 的所有元素，这些元素是 E 后面的兄弟节点之一
E.C	匹配标签名为 E 的所有元素，这些元素拥有 CSS 类名为 C，如果省略 E 则相当于*.C
E#I	匹配标签名为 E 的所有元素，这些元素的 id 特性值为 I，如果省略 E 则相当于*#I
E[A]	匹配标签名为 E 的所有元素，这些元素拥有特性 A
E[A=V]	匹配标签名为 E 的所有元素，这些元素的 A 特性值为 V
E[A^=V]	匹配标签名为 E 的所有元素，这些元素的 A 特性值以 V 开头
E[A$=V]	匹配标签名为 E 的所有元素，这些元素的 A 特性值以 V 结束
E[A!=V]	匹配标签名为 E 的所有元素，这些元素的 A 特性值不等于 V，或者根本就不存在 A 特性
E[A*=V]	匹配标签名为 E 的所有元素，这些元素的 A 特性值包含 V

掌握了这些知识后，回到选择器实验室页面，多花一些时间来练习表 2-2 中列出的各种类型的选择器。试着有针对性地选择元素，比如选择包含文本 Hello 或 Goodbye 的<span>元素（提示：你需要组合使用选择器来完成这个任务[1]）。

① 参考答案：div[title='myTitle1'] span,div[title='myTitle2'] span。

到目前为止，我们对选择器的强大功能探讨的还不够，其实还有更多的选项，可以更加出色地筛选页面元素。

2.1.4　通过位置选择元素

有时，需要根据元素在页面上的位置，或者相对于其他元素的位置来选择元素。可能要选择页面中的第一个链接、隔行段落，或者每个列表中的最后一个列表项。jQuery 支持实现这些特殊的选择机制。

比如下面这个选择器：

```
a:first
```

这种格式的选择器会匹配页面上的第一个<a>元素。

那么如何获取隔行元素呢？

```
p:odd
```

这个选择器会匹配所有奇数的段落元素。当然，我们也可以通过下面的代码匹配偶数的段落元素：

```
p:even
```

还有一种选择器：

```
ul li:last-child
```

选择父元素的最后一个子节点。在本例中，匹配的是每一个<ul>元素的最后一个<li>子节点。

这样的选择器有很多，有些通过 CSS 定义，有些是 jQuery 所特有的，它们为一些难题提供了令人惊叹的优雅解决方案。CSS 规范将这种类型的选择器称为伪类（pseudo-classes），但 jQuery 赋予了它新的名字——过滤器，因为这些选择器是用来过滤另外一个基础选择器的。这些过滤器非常容易识别，因为它们都以冒号（:）开头。记住，如果你省略了基础选择器，它就默认为*。

表 2-3 列出了这些位置过滤器（在 jQuery 文档中称为基础和子节点过滤器）。

表 2-3　jQuery 支持的位置过滤选择器

选　择　器	描　　述
:first	匹配上下文中的第一个元素。li a:first 返回列表项后代节点中的第一个链接
:last	匹配上下文中的最后一个元素。li a:last 返回列表项后代节点中的最后一个链接
:first-child	匹配上下文中的第一个子节点。li:first-child 返回每个列表中的第一个列表项
:last-child	匹配上下文中的最后一个子节点。li:last-child 返回每个列表中的最后一个列表项
:only-child	返回所有没有兄弟节点的元素
:nth-child(n)	匹配上下文中的第 *n* 个子节点。li:nth-child(2) 返回每个列表中的第二个列表项
:nth-child(even|odd)	匹配上下文中的偶数或奇数子节点。li:nth-child(even) 返回每个列表中的偶数列表项
:nth-child(Xn+Y)	匹配上下文中的由提供的公式计算出的子节点。如果 Y 是 0，则可以省略。li:nth-child(3n)返回每个列表中能被 3 整除的列表项，而 nth-child(5n+1)返回每个列表中所有能被 5 整除的列表项的下一个列表项
:even	匹配上下文中的偶数元素。li:even 返回所有偶数的列表项

（续）

选　择　器	描　　述
:odd	匹配上下文中的奇数元素。li:odd 返回所有奇数的列表项
:eq(*n*)	匹配第 *n* 个元素
:gt(*n*)	匹配第 *n* 个元素之后的元素（不包含第 *n* 个元素）
:lt(*n*)	匹配第 *n* 个元素之前的元素（不包含第 *n* 个元素）

这里有一个快捷技巧（总会有一些的，对吧？）。:nth-child 过滤器从 1 开始计数（为了与 CSS 兼容），而其他的选择器都是从 0 开始计数（遵循常用的编程约定）。可能刚开始这会让人有点困惑，不过多加练习后你就会习惯。

我们再深入研究一下。

考虑如下来自于实验室示例 DOM 的表格代码。它包含了几种编程语言以及这些语言的基本信息：

```
<table id="languages">
  <thead>
    <tr>
      <th>Language</th>
      <th>Type</th>
      <th>Invented</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Java</td>
      <td>Static</td>
      <td>1995</td>
    </tr>
    <tr>
      <td>Ruby</td>
      <td>Dynamic</td>
      <td>1993</td>
    </tr>
    <tr>
      <td>Smalltalk</td>
      <td>Dynamic</td>
      <td>1972</td>
    </tr>
    <tr>
      <td>C++</td>
      <td>Static</td>
      <td>1983</td>
    </tr>
  </tbody>
</table>
```

假如我们想获取表格中包含编程语言名称的所有单元格。因为它们都是每一行的第一个单元格，所以可以像下面这样写：

```
table#languages td:first-child
```

也可以这么写：

```
table#languages td:nth-child(1)
```

不过第一种语法更加简洁优雅。

为了获取语言类型的单元格，我们可以修改选择器的过滤器为:nth-child(2)，也可以使用:nth-child(3)或者:last-child 获取语言发明年份的单元格。如果只想获取表格的最后一个单元格（包含文本 1983 的单元格），可以使用 td:last。同样，td:eq(2)返回包含文本 1995 的单元格，而 td:nth-child(2)返回包含编程语言类型的所有单元格。再次强调，记住:eq 是从 0 开始计数的，而:nth-child 是从 1 开始计数的。

在继续学习之前，回到选择器实验室页面，尝试从列表中选择第 2 项和第 4 项[1]。然后尝试找到 3 种不同的方法，选择表格中包含文本 1972 的单元格[2]。此外，试着感受:nth-child 过滤器与绝对定位选择器之间的差异[3]。

尽管目前为止我们学到的 CSS 选择器非常强大，但还有更强大的 jQuery 选择器等待我们去探索。

2.1.5 使用CSS和自定义的jQuery过滤选择器

CSS 选择器赋予了我们强大的能力以及灵活性来匹配所需的 DOM 元素，不过还有一些选择器可以用来更进一步过滤选择项。

例如，当我们想选择所有处于选中状态的复选框。你可能会尝试以下代码：

```
$('input[type=checkbox][checked]')
```

但是，通过特性（checked）匹配只能检查控件在 HTML 标记中声明的控件*初始*状态[4]。而我们真正想要检查的是控件的实时状态。CSS 提供了伪类:checked，用来匹配处于选中状态的元素。比如，尽管 input 选择器选择的是所有<input>元素，但是 input:checked 能够将搜索范围缩小到只处于选中状态的<input>元素。

这似乎还不够，jQuery 还提供了许多强大的、不是通过 CSS 指定的自定义过滤选择器，可以更方便地标识目标元素。比如，自定义的:checkbox 选择器用来识别所有的复选框元素。这些自定义选择器组合起来将非常强大，思考下:checkbox:checked 和:radio:checked 所能选择的元素。

前面曾经讨论过，jQuery 在支持 CSS 过滤选择器的同时还定义了很多自定义选择器。表 2-4 列出了这些选择器。

---

① 参考答案：li:nth-child(2n)。

② 参考答案：tbody tr:eq(2) td:eq(2)、td:contains('1972')、tbody td:nth-child(3):eq(2)。

③ tbody td:nth-child(1)返回 Java、Ruby、Smalltalk、C++这 4 个单元格，而 tbody td:eq(1)仅仅返回描述 Java 语言特性的 Static 单元格。

④ 这里的描述是正确的，但是如果你在实验室页面测试这个选择器，会发现它能匹配到动态更新状态的复选框。这其实是 jQuery 1.4.2 的一个 BUG，我们需要使用 jQuery 的后续版本（比如 1.4.4）来获得正确的效果。

表 2-4　CSS 和自定义过滤器

选　择　器	描　　述	CSS 支持
:animated	选择处于动画状态的元素（第 5 章会介绍动画与特效）	
:button	选择按钮元素（包括 input[type=submit]、input[type=reset]、input[type=button]和 button）	
:checkbox	选择复选框元素（input[type=checkbox]）	
:checked	选择处于选中状态的复选框或单选按钮元素	√
:contains(food)	选择包含文本 food 的元素	
:disabled	选择处于禁用状态的元素	√
:enabled	选择处于启用状态的元素	√
:file	选择文件输入元素（input[type=file]）	
:has(selector)	选择至少包含一个匹配指定选择器的元素的元素	
:header	选择标题元素。比如<h1>到<h6>	
:hidden	选择隐藏元素	
:image	选择图片输入元素（input[type=image]）	
:input	选择表单元素（input、select、textarea、button）	
:not(selector)	选择不匹配指定选择器的元素	√
:parent	选择有子节点（包含文本）的元素[1]，空元素除外	
:password	选择口令元素（input[type=password]）	
:radio	选择单选框元素（input[type=radio]）	
:reset	选择重置按钮元素（input[type=reset]或者 button[type=reset]）	
:selected	选择列表中处于选中状态的<option>元素	
:submit	选择提交按钮元素（input[type=submit]或者 button[type=submit]）	
:text	选择文本输入框元素（input[type=text]）	
:visible	选择可见的元素	

很多 CSS 和自定义的 jQuery 过滤选择器是与表单相关的，允许优雅地指定特定元素类型或状态。我们也可以组合过滤选择器。比如，如果想只选择那些同时处于启用和选中状态的复选框，可以使用：

```
:checkbox:checked:enabled
```

在选择器实验室页面尽可能多尝试使用这些过滤器，直到你觉得自己能够很好地掌握它们的用法。

这些过滤器对于选择器集合来说是非常有益的补充，然而反转这些过滤器又会怎样呢？

1. 使用**:not** 过滤器

如果想对选择器取反，假设要匹配所有不是复选框的<input>元素，可以使用:not 过滤器。

① :parent 是选择器:empty 的取反。

例如，选择非复选框的<input>元素，可以使用：

```
input:not(:checkbox)
```

但是要小心！这很容易使你误入歧途并且会得到意料之外的结果！

比如，假设我们想选择所有的图片，但是要排除 src 特性值包含 dog 文本的图片。我们可以快速地构造出下面的选择器：

```
$(':not(img[src*="dog"])')
```

但是如果使用了这个选择器，就会发现不仅选择了所有 src 特性值不包含 dog 的图片，而且也选择了 DOM 中所有不是图片的元素！

哎呀！想起来了，如果省略了基础选择器，它默认就是*，因此我们这个错误的选择器的意思是“获取所有元素，排除那些 src 特性值包含 dog 的图片元素”。我们真正想要的是“获取所有图片元素，排除那些 src 特性包含 dog 的图片”，正确的表达式是下面这样：

```
$('img:not([src*="dog"])')
```

再次强调下，使用实验室页面多做练习，直到你熟练使用:not 过滤器来反转选择器。

jQuery 还添加了一个自定义过滤器，用来帮助我们通过元素的父子关系来选择元素。

警告　如果你仍在使用 jQuery 1.2，注意类似于:not()和:has()这样的过滤选择器只能接受其他过滤选择器。不能向它们传递包含元素表达式的选择器。这个限制在 jQuery 1.3 中已经消失了。

---

2. 使用:has 过滤器

如前所述，CSS 定义了一个有用的选择器，用来选择特定父节点的子节点。比如下面这个选择器：

```
div span
```

会选择<div>元素子节点中所有的<span>元素。

但是如果想要取反呢？如果想选择包含<span>元素的所有<div>元素，该怎么办呢？

这就是:has()过滤器的职责所在。考虑如下选择器：

```
div:has(span)
```

它会选择<div>祖先元素，而不是<span>子孙元素。

当我们想要选择结构复杂的元素时，这会是一个强大的处理机制。比如，假设我们想要找出包含某种图片元素的表格行，这种图片元素可以使用 src 特性来唯一标识。可以使用如下选择器，

```
$('tr:has(img[src$="puppy.png"])')
```

这会返回在子节点层次结构中包含已标识图片的任何表格行元素。

可以肯定的是，在以后将要探讨的代码中，这个过滤器和其他的 jQuery 过滤器会共同扮演重要角色。

如前所述，jQuery 提供了庞大的工具集用来选择页面的现有元素，以便使用 jQuery 方法来

进行操作，我们会在第 3 章中对此进行详细讨论。在学习这些操作方法之前，先来看下如何使用`$()`函数来创建新的 HTML 元素。

2.2 创建新的 HTML

有时，我们需要生成新的 HTML 代码以插入到页面中。这些动态创建的元素可能很简单，比如在一定条件下显示一些附加文本，也可以复杂，比如从服务器数据库获取数据，再用这些数据生成表格。

有了 jQuery，创建动态元素就是小菜一碟，和我们在第 1 章中介绍的一样，因为`$()`函数除了能够选择页面中的现有元素，还可以从 HTML 字符串创建元素。考虑以下代码：

```
$("<div>Hello</div>")
```

这个表达式创建了一个要添加到页面中的新的`<div>`元素。可以在由页面现有元素组成的包装集上运行任何 jQuery 方法，也可以在新创建的 HTML 片段上运行。初看起来这似乎并不吸引人，但是当把事件处理程序、Ajax 以及特效揉合在一起综合考虑（我们将在接下来的几章中学习），你就会发现这非常方便。

注意，如果想创建一个空白`<div>`元素，就可以使用下面这样的快捷方式：

```
$("<div>")
```

这与`$("<div></div>")`和`$("<div/>")`是等价的，不过我建议使用符合标准的标记，这些标记将对任何可以包含其他元素的元素类型书写完整的开始和结束标签。

创建这种简单的 HTML 元素非常容易，利用 jQuery 方法的链式操作，创建更复杂的元素也不会太难。我们可以将任何 jQuery 方法应用到包含新创建元素的包装集。比如，我们可以通过`css()`方法为元素应用样式，也可以通过`attr()`方法为元素创建特性，不过 jQuery 提供了更好的方法来完成这一任务。

我们可以传递第二个参数到创建元素的`$()`方法中，用来指定特性以及特性值。这个参数是一个 JavaScript 对象，它的属性作为特性的名称和值来应用到元素上。

假设我们想创建一个拥有很多特性的图片元素，它拥有某种样式，另外还要让它响应单击事件。参见代码清单 2-1 中的代码。

代码清单 2-1 动态创建一个全功能的`<img>`元素

```
$('<img>',                                  ❶ 创建基本的<img>元素
  {
    src: 'images/little.bear.png',          ❷ 设置各种特性
    alt: 'Little Bear',
    title:'I woof in your general direction',
    click: function(){                      ❸ 设置单击事件处理函数
      alert($(this).attr('title'));
    }
  })
  .css({                                    ❹ 为图片添加样式
    cursor: 'pointer',
```

```
    border: '1px solid black',
    padding: '12px 12px 20px 12px',
    backgroundColor: 'white'
  })
  .appendTo('body');
```

❺ 将此元素添加到页面中

代码清单 2-1 的 jQuery 单行表达式首先创建一个基本的<img>元素❶，赋予它一些重要的特性，比如图片来源、替换文字、悬停文字❷，然后添加样式使其看起来像打印出来的照片❹，最后把它附加到 DOM 树中❺。

我们也在代码中玩了点花样，使用 attribute 对象创建了一个事件处理函数，用来弹出对话框（显示图片的提示文字）以响应图片的单击事件❸。

jQuery 不仅可以在 attribute 参数中指定特性，而且可以为所有类型的事件创建处理函数（我们会在第 4 章中深入讨论），还可以向少数 jQuery 方法传递值，这些方法的目的是为了设置元素各个方面的参数。虽然我们还没有学习这些方法，不过可以先为以下方法指定值（在下一章中会讨论大部分方法）：val、css、html、text、data、width、height 以及 offset。

因此，在代码清单 2-1 中，我们可以省略对链式方法 css()的调用，用 attribute 参数中对象的属性来代替，如下所示：

```
css: {
  cursor: 'pointer',
  border: '1px solid black',
  padding: '12px 12px 20px 12px',
  backgroundColor: 'white'
}
```

不管我们如何重构代码，它看起来还是有点笨重（为了方便阅读，我们分多行来显示代码，并且进行了逻辑缩进），不过它的确实现了很多功能。这样的语句在使用 jQuery 的页面中会很常见，如果你觉着有点不知所措，不用担心，接下来的几章会介绍这个语句的每一个方法。不久，你将会习惯编写这样的复合语句了。

图 2-5 显示了这段代码的执行结果，包括页面首次加载完毕后的效果（参见图 2-5a）和单击图片后的效果（参见图 2-5b）。

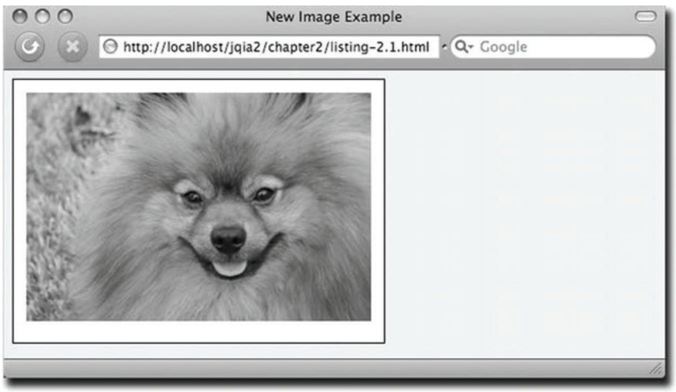


图 2-5a 动态创建复杂的元素，包括这张单击后会弹出对话框的图片，简直易如反掌

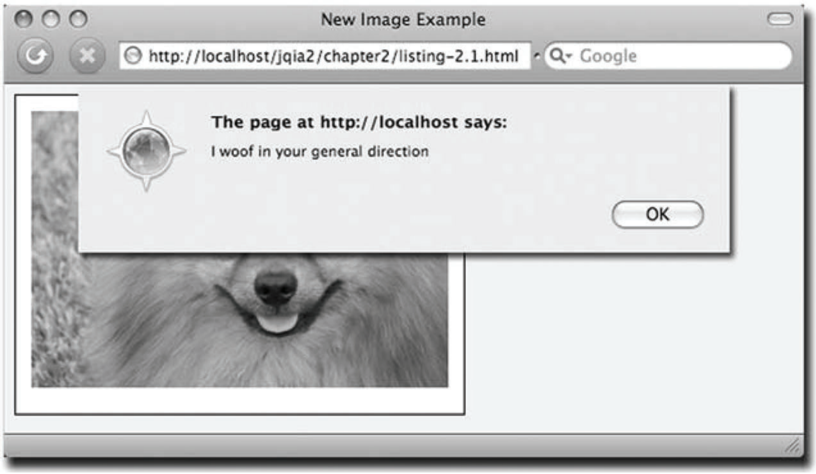


图 2-5b　动态创建的图片拥有所有需要的样式和特性，包括为响应鼠标单击行为而弹出的对话框

本例的完整代码可以从本书的项目代码中找到：chapter2/listing-2.1.html。

到目前为止，我们已经将包装器方法应用到了整个包装集上，这些包装集是通过向 jQuery 函数传递选择器来创建的。但有时候在应用包装器方法之前，我们可能想对包装集进行更进一步的操作。

2.3　管理包装集

一旦获取了一个包装集，不管是通过选择器从现有的 DOM 元素中识别的元素，还是通过 HTML 片段创建的新元素（或者两者组合），都可以使用强大的 jQuery 方法集对其进行操作。我们会在下一章讨论这些方法。但是如果想进一步精简由 jQuery 函数创建的包装集，该怎么办呢？本节中，我们会探索很多这样的方法，用来提取、扩展或者获取包装集的子集，以便对其进行操作。

为了帮助你学习这方面内容，本章的项目代码还包含了另一个实验室页面：jQuery 操作实验室页面（chapter2/lab.operations.html）。这个页面和本章前面介绍的选择器实验室页面很相似，如图 2-6 所示。

这个新的实验室页面不仅看起来像选择器实验室页面，而且操作方式也很类似。不过，在这个页面中，不能输入选择器，而是要输入能生成包装集的完整 jQuery 操作代码。这个操作会在 DOM Sample 的上下文中执行，并且像选择器实验室页面那样显示操作结果。

从某种意义上说，jQuery 操作实验室页面是选择器实验室页面更通用版。后者只允许我们输入单个的选择器，而 jQuery 操作实验室页面允许输入任何可以生成 jQuery 包装集的表达式。由于 jQuery 链的工作方式，这个表达式可以包含多个包装器方法，这使得此页面成为研究 jQuery 操作的强大实验室。

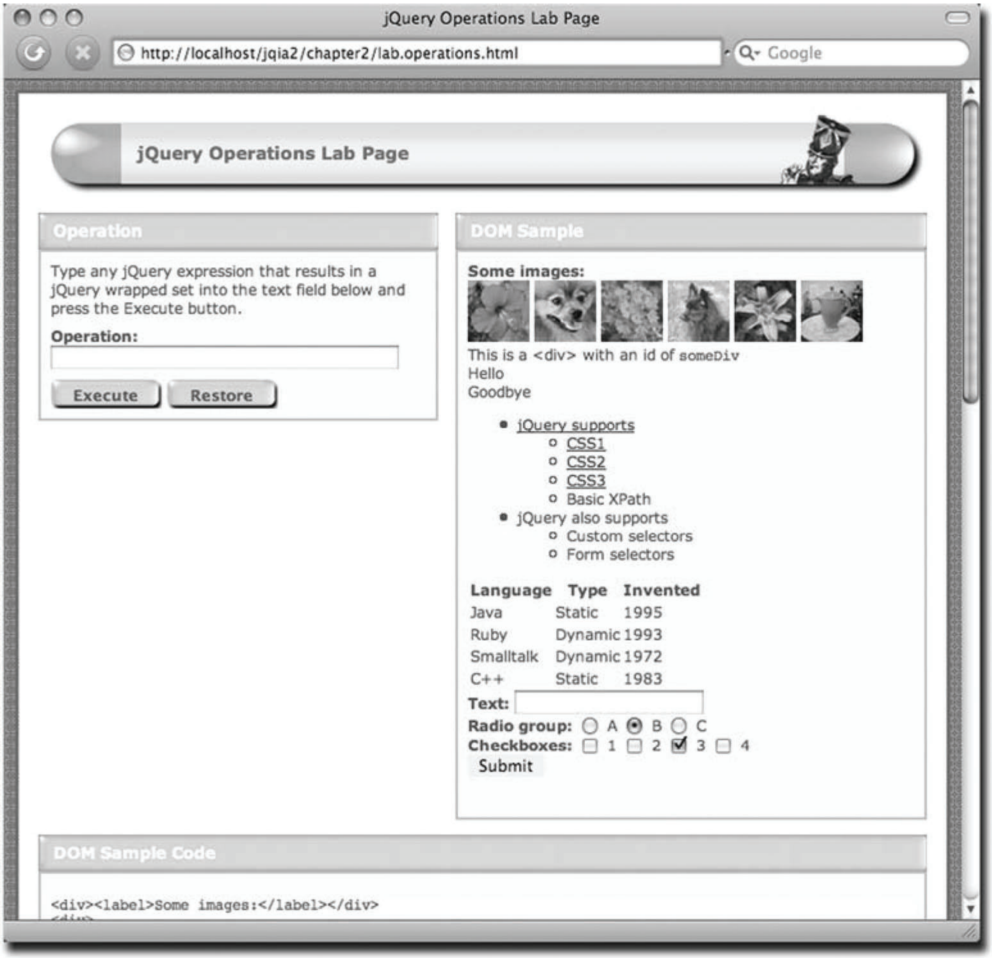


图 2-6 jQuery 操作实验室页面允许我们实时构建包装集，以便理解如何创建和管理包装集

注意，你需要输入正确的语法，以及能够产生 jQuery 包装集的表达式。否则，你会看到一些毫无用处的 JavaScript 错误。

为了获得更直观的感受，在浏览器中打开此页面并在操作框中输入下面的代码：

```
$('img').hide()
```

然后单击 Execute 按钮。

这个操作在 DOM Sample 的上下文中执行，你可以看到所有图片从示例页面上消失了。无论进行了什么操作，你都可以通过单击 Restore 按钮将 DOM Sample 恢复到初始状态。

在接下来几节中，我们将看到这个新的实验室页面的作用。在后面几章中需要测试各种 jQuery 操作的时候，此页面也会非常有用。

2.3.1　确定包装集的大小

前面提到过，jQuery 包装元素集的运行方式与数组非常相似。像 JavaScript 数组那样，这个伪数组有一个包含包装元素个数的 length 属性。

如果想使用方法而不是 length 属性，可以用 jQuery 定义的 size()方法，它也能返回相同的信息。

考虑以下语句：

```
$('#someDiv')
  .html('There are '+$('a').size()+' link(s) on this page.');
```

嵌入到此代码的 jQuery 表达式匹配所有的<a>元素，并使用 size()方法返回匹配元素的个数。用此个数来拼接文本字符串，然后调用 html()方法（我们会在下一章学习），把这个文本字符串设置为 id 为 someDiv 的元素的内容。

size()方法的正式语法如下所示。

方法语法：size

size()

返回包装集中元素的个数

参数

无

返回值

元素的个数

好了，现在我们知道匹配了多少个元素。那么如何直接访问它们呢？

2.3.2　从包装集中获取元素

一般来说，一旦获取了元素包装集，就可以使用 jQuery 方法对其执行某种操作。比如，调用 hide()方法隐藏全部元素。但有时候，我们希望获取其中一个或者多个元素的直接引用，以便对其执行一些原始的 JavaScript 操作。下面来看看 jQuery 提供了哪些方法来帮助我们完成此任务。

1. 通过索引获取元素

因为 jQuery 允许我们将包装集当成 JavaScript 数组，所以我们可以使用简单的数组下标，即通过位置来获取包装序列中的任何元素。比如，从页面上带有 alt 特性的所有<img>元素的包装集中获取第一个元素，可以使用如下语句：

```
var imgElement = $('img[alt]')[0]
```

如果你更喜欢使用方法而不是数组下标，那可以使用 jQuery 定义的 get()方法。

方法语法：get

get(index)

获取包装集中的一个或全部匹配元素。如果不指定参数，则包装集中的所有元素就以 JavaScript 数组形式返回。如果提供了 index 参数，则会返回 index 所对应的元素

参数

index （数值）需要返回的单个元素的下标。如果省略，则整个集合会以数组形式返回

返回值

一个 DOM 元素或 DOM 元素数组

代码片段：

```
var imgElement = $('img[alt]').get(0)
```

和前面使用了数组下标的示例是等价的。

get()方法也可以接受负的下标值作为参数。这种情况下，它从包装集的末尾开始查找元素。比如，.get(-1)返回包装集中的最后一个元素，.get(-2)返回倒数第二个元素，依此类推。

除了获取单个元素，get()还可以返回一个数组。

虽然我们推荐使用 toArray()方法（下节将会讨论）获取包装集中元素的 Javascript 数组，但 get()方法也可以用来获取由所有包装元素组成的普通 JavaScript 数组。

上述获取数组的方式是为了向后兼容 jQuery 以前的版本。

get()方法返回的是一个 DOM 元素，然而有时候我们希望得到的是包含指定元素的包装集，而不是元素本身。下面的代码看起来有点奇怪：

```
$($('p').get(23))
```

因此，jQuery 提供了 eq()方法，用来模仿:eq 过滤器的行为。

方法语法：eq

eq(index)

获取包装集中与 index 参数相对应的元素，并返回只包含此元素的新包装集

参数

index （数值）需要返回的单个元素的下标。和 get()方法一样，负的下标值可以指定从集合的末尾开始查找元素

返回值

包含一个或零个元素的包装集

获取包装集中第一个元素的操作是很常见的，因此有更便捷的方法来简化这个操作，即 first()方法。

方法语法：first

first()
获取包装集中的第一个元素，并返回只包含此元素的新包装集。如果原包装集为空，则返回空包装集

参数

无

返回值

包含一个或零个元素的包装集

如你所料，有对应的方法来获取包装集中的最后一个元素。

方法语法：last

last()
获取包装集中的最后一个元素，并返回只包含此元素的新包装集。如果原包装集为空，则返回空包装集

参数

无

返回值

包含一个或零个元素的包装集

现在来探讨下获取由包装集元素组成的数组的推荐方法。

2. 以数组形式获取所有元素

如果你想以一个 DOM 元素的 JavaScript 数组的形式来获取包装集里的所有元素，那么可以使用 jQuery 提供的 `toArray()`方法。

方法语法：toArray

toArray ()
将包装集里的所有元素作为 DOM 元素数组返回

参数

无

返回值

由包装集中的 DOM 元素组成的 JavaScript 数组

考虑下面这个例子：

```
var allLabeledButtons = $('label+button').toArray();
```

这条语句会收集页面上`<label>`元素后面同级节点的所有`<button>`元素，将它们封装成

jQuery 包装器，然后创建由这些<button>元素组成的 JavaScript 数组，将其赋值给 allLabeled-Buttons 变量。

3. 获取元素的索引

get()通过给定下标来查找元素，我们还可以使用它的逆操作 index()，来获取包装集中特定元素的下标。index()方法的语法如下所示。

2

方法语法：index

index (element)

在包装集中查找传入的元素，返回它在包装集中的下标，或者返回包装集中的第一个元素在同级节点中的下标。如果没有找到此元素，则返回–1

参数

element	（元素|选择器）需要获取下标的元素引用，或者用来识别元素的选择器。如果省略此参数，则找出包装集中的第一个元素在同级节点中的下标

返回值

传入的元素在包装集或者在其同级节点中的下标，若找不到则返回–1

假设由于某些原因，我们想要知道 id 为 findMe 的图片在整个页面图片集合中的下标号。可以通过下面语句来获取这个值：

```
var n = $('img').index($('img#findMe')[0]);
```

也可以简写为：

```
var n = $('img').index('img#findMe');
```

index()方法也可以用来查找一个元素在其父节点里的下标值（也就是在其同级节点中的位置）。比如：

```
var n = $('img').index();
```

这会将 n 设置为第一个<img>元素在其父节点里的下标号。

现在，如何来调整包装元素集而不是获取元素的直接引用或者它们的下标呢？

2.3.3 分解元素包装集

一旦获得了一个元素包装集，你可能想向其中添加元素来扩充包装集，或者将包装集缩减到由原始匹配元素组成的子集。jQuery 提供了一整套管理包装集的方法。首先来看下如何添加元素到包装集。

1. 添加更多元素到包装集

我们经常需要添加更多的元素到现有的包装集。当我们对原始包装集应用了一些方法后，再试图向其中添加更多的元素时，这个功能极其有用。记住，jQuery 链可以在单个表达式中执行大量任务。

我们马上就会看到此类场景下的一些具体示例，不过先从简单的情况开始。假设想要匹配带有 alt 或 title 特性的所有<img>元素。强大的 jQuery 选择器允许使用单个选择器完成这一任务，例如：

```
$('img[alt],img[title]')
```

但为了说明 add()方法的操作，我们通过下面的语句来匹配相同的集合：

```
$('img[alt]').add('img[title]')
```

以这种方式调用 add()方法，可以将多个选择器链接起来，创建满足任何一个选择器的元素组合。

在 jQuery 方法链里，类似 add()的方法也是非常重要的（并且比聚合选择器[1]更灵活），因为它们没有扩充原始的包装集，而是创建一个新的包装集来保存结果。我们很快就会看到当 add()方法和其他方法[比如用来“收回”对包装集扩充操作的 end()方法（参见 2.3.6 节）]一起使用时，它是如何发挥强大作用的。

下面是 add()方法的语法。

方法语法：add

add(expression,context)

创建包装集的副本并向其中添加由 expression 参数指定的元素。expression 可以是选择器、HTML 片段、DOM 元素或 DOM 元素数组

参数

expression	（选择器|元素|数组）指定要添加到包装集的元素。该参数如果是 jQuery 选择器，则将全部匹配的元素添加到包装集中。如果该参数是 HTML 片段，则创建相应的元素并添加到包装集中。如果参数是 DOM 元素或 DOM 元素数组，则直接将其添加到包装集中
context	（选择器|元素|数组|jQuery）指定一个上下文，用来缩小匹配第一个参数的元素的查找范围。这和传递到 jQuery()函数中的上下文参数是一样的。2.1.1 节中有对该参数的详细描述

返回值

所添加元素的原始包装集副本

在浏览器中打开 jQuery 操作实验室页面，输入以下表达式：

```
$('img[alt]').add('img[title]')
```

然后单击 Execute 按钮。这会执行上面的 jQuery 操作，选中拥有 alt 或 title 特性的所有图片。

检查 DOM Sample 的 HTML 源代码，可以发现所有花朵图片都拥有 alt 特性，小狗图片拥有 title 特性，而咖啡壶图片两个特性都没有。因此，应该料到除了咖啡壶图片外其他图片元素都将成为包装集的一员。图 2-7 展示了显示结果的屏幕截图。

① 在 2.1.2 中有介绍，通过逗号操作符将多个选择器合并成一个选择器。

可以看到，6 张图片中的 5 张（除咖啡壶图片以外）添加到包装集中。由于本书的印刷版里图像是灰度图，因此其中红色的轮廓线可能不太清晰，不过如果你已经下载了示例代码（你应该已经下载了）并且跟着示例代码来学习（这是推荐的学习方式），就可以清楚地看到了。

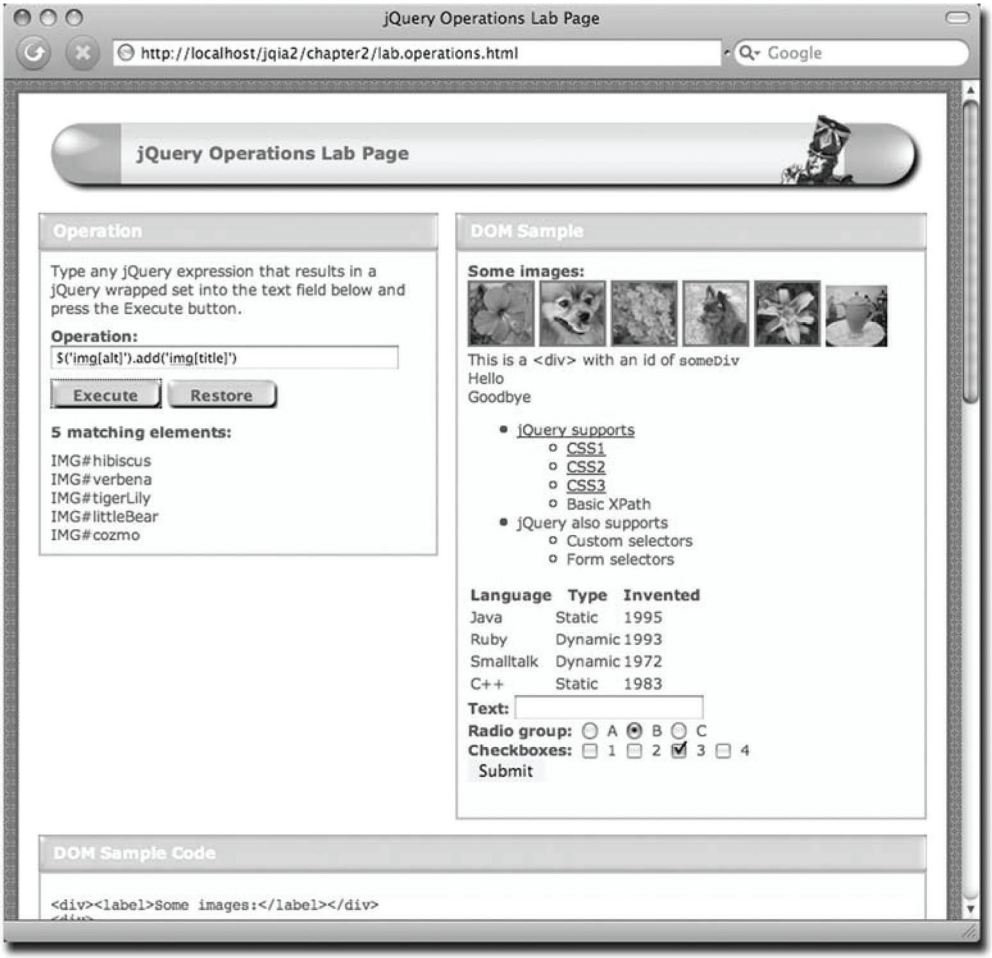


图 2-7　不出所料，jQuery 表达式匹配了拥有 `alt` 或 `title` 特性的图片元素

接下来看看 `add()`方法更实际的一个应用。假设想对所有拥有 `alt` 特性的`<img>`元素应用粗边框，然后对所有拥有 `alt` 或 `title` 特性的`<img>`元素应用一定程度的透明效果。而这回，CSS 选择器中的逗号操作符（，）却无能为力，因为我们要把操作应用到包装集，然后添加更多的元素到该包装集中，之后再应用另一个操作。可以使用多条语句轻松完成这一任务，但利用 jQuery 链的强大能力，我们就可以高效且优雅地在单个语句中完成这一任务，如下所示：

```
$('img[alt]')
  .addClass('thickBorder')
```

```
    .add('img[title]')
    .addClass('seeThrough')
```

在这个语句中，我们首先创建了一个由拥有 alt 特性的所有<img>元素组成的包装集，然后应用预定义 CSS 类 thickBorder 实现粗边框，添加带有 title 特性的<img>元素，最后对新扩充的包装集应用 CSS 类 SeeThrousk 实现透明效果。

把这条语句输入到 jQuery 操作实验室页面（已经预定义了要引用的 CSS 类），结果如图 2-8 所示。

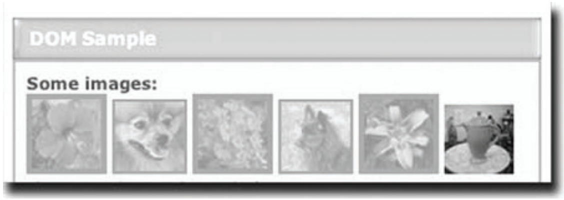


图 2-8　通过 jQuery 链，在一条单个语句中完成复杂操作的结果

可以看到，在这些结果中花朵图片（拥有 alt 特性）有粗边框，而除咖啡壶（即没有 alt 又没有 title 特性的图片）之外的其他所有图片都褪色了，这是因为应用了 CSS 的透明度规则。

只要给定元素的直接引用，也可以使用 add()方法来将这些元素添加到现有包装集中。把元素引用或者由元素引用组成的数组传递给 add()方法，就可以将这些元素添加到包装集。如果已经有一个变量名为 someElement 的元素引用，则可以将它添加到带有 alt 特性的所有图片包装集中：

```
$('img[alt]').add(someElement)
```

这似乎还不够灵活，add()方法不仅允许把现有元素添加到包装集，而且还可以通过传递一个包含 HTML 标记的字符串来添加新创建的元素。考虑如下代码：

```
$('p').add('<div>Hi there!</div>')
```

这个代码片段创建文档中所有<p>元素的包装集，然后新建一个<div>元素并将其添加到该包装集中。注意，这么做只是把新创建的元素添加到包装集中，并没有把此元素添加到 DOM 树中。这时可以使用 jQuery 的 appendTo()方法（耐心点，很快就会讨论到这些方法）将选中的元素以及新创建的 HTML 添加到 DOM 树的某个部分。

调用 add()扩充包装集简单而有力，现在来看下从包装集中删除元素的 jQuery 方法。

2. 整理包装集中的内容

我们已经看到使用 add()方法将多个选择器链接起来，从而扩充包装集，这一点儿也不困难。同样也可以使用 not()方法将多个选择器链接起来以形成差集关系。这和前面介绍的:not 过滤器类似，但它可以采用类似 add()的方式，在 jQuery 方法链里的任意位置删除包装集中的元素。

假设我们想选择页面上拥有 title 特性的所有<img>元素，除了那些 title 特性值中包含 puppy 文本的图片。单个的选择器就可以处理这种情况（img[title]:not([title*=puppy])），但是为了说明，我们假装忘记:not()过滤器的存在。通过调用 not()方法，从包装集中删除与

选择器相匹配的元素，就可以描述差集关系。要完成上述匹配，可以这样写：

```
$('img[title]').not('[title*=puppy]')
```

在 jQuery 操作实验室页面输入该表达式，然后运行它。你将看到只有棕色小狗图片高亮显示。而黑色小狗图片虽然由于拥有 title 特性而存在于原始的包装集中，但却在调用 not()后被删除了，因为它的 title 特性包含文本 puppy。

方法语法：not

not(expression)

创建包装集的副本，从中删除那些与 expression 参数值指定的标准相匹配的元素

参数

expression	（选择器|元素|数组|函数）指定要删除的元素。如果该参数是 jQuery 选择器，则删除任何匹配 expression 的元素。如果传递的是元素或者元素数组，则删除包装集中的这些元素 如果传递的是函数，则会为包装集中的每个元素调用此函数（this 指定当前元素），并从包装集中删除调用的返回值为 true 的元素

返回值

不含已删除元素的原始包装集副本

可以向 not()方法传递元素引用或者元素数组引用来删除单个元素。而后者会非常有趣而强大，这是因为任何 jQuery 包装集都可以作为由元素引用组成的数组。

当需要最大限度的灵活性时，可以向 not()传递一个函数，这样就可以逐个决定是否保留此元素。考虑如下示例：

```
$('img').not(function(){ return !$(this).hasClass('keepMe'); })
```

这个表达式会选择所有<img>元素，然后删除不包含 CSS 类 keepMe 的元素。

当难以使用过滤器来过滤包装集时，可以通过编写代码来过滤包装集里的元素。

有时，我们希望传递到 not()中的函数表达相反的意思，此时可以使用 not()的反方法 filter()，它以类似的方式工作，不过如果元素的函数返回 false 就删除该元素。

比如，假设我们希望创建一个包装集，这个包装集由包含数字值的所有<td>元素组成。虽然 jQuery 选择器的表达式非常强大，但还是无法完成这个特殊的需求。在这种情况下，filter()方法就可以派上用场了，如下所示：

```
$('td').filter(function(){return this.innerHTML.match(/^\d+$/)})
```

这个表达式创建一个由所有<td>元素组成的包装集，然后对包装集中的每个元素调用传递到 filter()方法中的函数（当前匹配的元素作为调用的 this 值）。该函数使用正则表达式来决定元素内容是否匹配指定的模式（一个或多个整数序列），如果不匹配则返回 false。过滤器函数调用返回 false 的任何元素都不会包含在返回的包装集中。

方法语法：filter

filter(expression)

创建包装集的副本，并从中删除与 expression 参数值指定的标准不匹配的元素集合

参数

expression	（选择器|元素|数组|函数）指定要删除的元素。如果参数是 jQuery 选择器，则删除所有不匹配 expression 的元素 如果传递的是元素或者元素数组，则删除包装集中除了这些元素之外的所有元素 如果传递的是函数，则会对包装集的每个元素分别调用此函数（this 指定当前元素），并从包装集中删除函数调用返回值为 false 的元素

返回值

不包含已过滤元素的原始包装集副本

再次打开 jQuery 操作实验室页面，输入上面的表达式并执行。你将会看到 Invented 那一列的表格单元里只有<td>元素被选中了。

在调用 filter()方法时，也可以传入选择器表达式。当以这种方式使用时，filter()方法和相应的 not()方法的操作方式相反，即删除所有不匹配选择器的元素。filter()并不是个超级强大的方法，因为如果使用更富限制性的选择器，通常会更容易些，但是在 jQuery 方法链中它却非常有用。例如，考虑如下代码：

```
$('img')
  .addClass('seeThrough')
  .filter('[title*=dog]')
  .addClass('thickBorder')
```

这个链式语句选择所有图片，并对所有图片应用 CSS 类 seeThrough，然后只保留集合中 title 特性包含 dog 文本的图片元素，最后再对余下的图片元素应用另一个名为 thickBorder 的类。结果是所有的图片都变成半透明的，但是只有棕色小狗图片用粗边框围了起来。

not()和 filter()方法给了我们强大的处理手段，可以根据任何有关包装元素的标准来动态地调整元素包装集。还可以根据元素在包装集中的位置，获得其子集。下面接着学习这些方法。

3. 获取包装集的子集

有时，你可能需要通过元素在包装集中的位置来获取包装集的子集。jQuery 提供了 slice()方法来完成此任务。这个方法创建并返回由原始包装集的任意连续部分（或者切片）组成的新包装集。

如果我们想获取一个包装集，它包含来自于另一个包装集的一个单个元素，并且基于该元素在原始包装集中的位置，那么就可以使用 slice()方法，通过传递原始包装集中元素的位置（从 0 开始计数）来创建此包装集。

比如，为了获取第 3 个元素，可以这么写：

```
$('*').slice(2,3);
```

方法语法：slice

slice(begin,end)

创建并返回新包装集，此包装集包含匹配集中一个连续的部分

参数

begin	（数字）在返回的切片中第一个元素的位置，从 0 开始计数
end	（数字）不包含在切片中的第一个元素的位置[1]（可选的），或者说是切片中最后一个元素的下一个元素的位置，从 0 开始计数。如果省略，则切片会扩展到包装集的最后一个元素

返回值

新建的包装集

这个语句选择页面上的所有元素，然后生成新包装集，它包含从匹配的元素集合中获取的第 3 个元素。

注意，这和`$('*').get(2)`是不同的，后者返回包装集中的第 3 个元素，而不是包含此元素的包装集。

因此，下面这个语句

```
$('*').slice(0,4);
```

选择页面上的所有元素，然后创建包含前 4 个元素的新包装集。

要获取包装集尾部的元素，使用下面这个语句：

```
$('*').slice(4);
```

匹配页面上的所有元素，然后返回一个不包含前 4 个元素的新包装集。

另一个可以用来获取包装集子集的方法是 `has()`。类似于`:has` 过滤器，这个方法测试包装元素的子节点，并使用这个测试条件选择将要组成子集的元素。

方法语法：has

has(test)

创建并返回新包装集，此包装集只包含原始包装集中子节点匹配 test 表达式的元素

参数

test	（选择器|元素）要应用到包装元素所有子节点上的选择器或是要测试的元素。只有特定的元素会包含在返回的包装集中，这些元素至少包含一个匹配选择器的子节点，或者其子节点就是传递的元素参数 test

返回值

结果包装集

① 此位置在开始位置（begin）之后。

比如，考虑这行代码：

```
$('div').has('img[alt]')
```

这个表达式会创建包含所有<div>元素的包装集，然后创建并返回包含特定<div>元素的第 2 个集合，这些<div>元素至少包含一个带有 alt 特性的<img>元素。

4. 转换包装集中的元素

我们经常需要对包装集中的元素执行转换操作。比如，要收集包装集中每个元素的 id 值该怎么做？或者如何从表单元素组成的包装集中收集值，以便从中生成查询字符串呢？在这种情况下 map()方法就派上用场了。

方法语法：map

map(callback)

为包装集中的每一个元素调用回调函数，并将返回值收集到 jQuery 对象实例中

参数

callback	（函数）回调函数，为包装集中的每个元素调用该函数。此函数接受两个参数：元素在集合中的下标（从 0 开始计数），以及元素本身。当前元素也被作为函数的上下文（this 关键字）

返回值

由已转换值组成的包装集

比如，下面的代码会将页面上所有<div>元素的 id 值收集到一个 JavaScript 数组中：

```
var allIds = $('div').map(function(){
  return (this.id==undefined) ? null : this.id;
}).get();
```

如果回调函数的某个调用返回 null，则相应的条目不会包含在最终返回的集合中。

5. 遍历包装集中的元素

为了收集元素的值或者按照其他方式转换元素，我们可以使用 map()方法遍历包装集中的元素，但是在许多情况下需要以更加通用的目的来遍历元素。在下面这些场合中，jQuery 的 each()方法凸显了它的价值。

此方法的一个用例是，使用它来设置匹配集中所有元素的属性值。比如，考虑这段代码：

```
$('img').each(function(n){
  this.alt='This is image['+n+'] with an id of '+this.id;
});
```

这个语句为页面上的每个图片元素调用传入的函数，使用元素的下标值和 id 值修改其 alt 属性。

方法语法：each

each(iterator)

遍历匹配集里所有的元素，为每一个元素调用传入的迭代函数

参数

iterator	（函数）回调函数，为匹配集中的每个元素调用。此函数接受两个参数：元素在集合中的下标（从 0 开始计数）以及元素本身。当前元素也被作为函数的上下文（this 引用）

返回值

包装集

为了方便起见，each()方法也可以用来遍历 JavaScript 数组对象甚至单个对象（不是后来介绍的拥有很多实用方法的对象[①]）。考虑下面这个示例：

```
$([1,2,3]).each(function(){ alert(this); });
```

这个语句会为传入$()中数组的每个元素调用迭代函数，函数中的 this 指向单独的数组项。

还没有结束呢！利用 jQuery，我们可以根据包装集元素相对于 DOM 中其他元素的关系来获取子包装集的能力。下面就来看看这是怎么做到的。

2.3.4 使用关系获取包装集

jQuery 允许根据包装元素相对于 HTML DOM 中其他元素的层次关系，从现有集合中获取新的包装集。

表 2-5 展示了这些方法以及关于它们的描述。这些方法大都接受一个可选的选择器表达式参数，可以用来选择要收集到新集合中的元素。如果没有传入选择器参数，则会选择所有符合条件的元素。

除了 contents()和 offsetParent()两个方法外，表 2-5 中的所有方法都可以接受选择器字符串参数，用来对结果进行过滤。

表2-5 根据与HTML DOM中其他元素的关系获取新包装集的方法

方　法	描　述
children([selector])	返回由每个包装元素所有的子节点[②]（不包含重复的子节点）组成的包装集
closest([selector])[③]	返回由与传入参数匹配的单个邻近祖先元素组成的包装集
contents()	返回由每个元素的内容组成的包装集，包括文本节点（这个方法常用来获取<iframe>元素的内容）
next([selector])	返回由每个包装元素后面下一个同级元素（不包含重复元素）组成的包装集

① 这里指的是通过$()函数创建的 jQuery 包装器对象。

② children()方法在直接子节点中查找匹配元素，类似的 find()方法则在所有的后代节点中查找元素。

③ 这里的描述不准确，参数 selector 是必须的，否则.closest()返回的永远是空的 jQuery 对象。

（续）

方　　法	描　　述
nextAll([selector])	返回由每个包装元素后面所有的同级元素组成的包装集
nextUntil([selector])	返回由每个包装元素后面所有的同级元素组成的包装集，直至遇到与选择器相匹配的元素，但不包括此元素。如果选择器没有匹配任何元素，或者省略了选择器参数，则会选择后面所有的同级元素
offsetParent()	返回由包装集中离第一个元素最近的，使用相对或者绝对定位的祖先元素组成的包装集
parent([selector])	返回由每个包装元素的直接父元素（不包含重复元素）组成的包装集
parents([selector])①	返回由每个包装元素所有的祖先元素（不包含重复元素）组成的包装集。这不仅包括直接父元素，还包含其上的所有祖先元素，但是不包括文档根节点
parentsUntil([selector])	返回由每个包装元素所有的祖先元素组成的包装集，直至遇到选择器匹配的元素，但不包括此元素。如果选择器没有匹配到元素，或者是省略了选择器参数，则选择所有的祖先元素
prev([selector])	返回由每个包装元素前面紧邻的同级元素（不包含重复元素）组成的包装集
prevAll([selector])	返回由每个包装元素前面所有的同级元素组成的包装集
prevUntil([selector])	返回由每个包装元素前面所有的同级元素组成的包装集，直至遇到选择器匹配的元素，但不包括此元素。如果选择器没有匹配到元素，或者是省略了选择器参数，则选择后面所有的兄弟元素
siblings([selector])	返回由每个包装元素的所有同级元素（不包含重复元素）组成的包装集

考虑一个情景，按钮的事件处理函数（详见第 4 章）触发时，处理器内部的 this 关键字指向按钮元素。假设我们想找出此按钮是在哪个<div>元素块中定义的，用 closest()方法就可以轻松搞定：

```
$(this).closest('div')
```

但是这只能查找最近的<div>祖先元素。如果我们要找的<div>在更高层次的祖先元素中该怎么办呢？没问题。我们可以调整传入 closest()的选择器参数来区分要选择的元素：

```
$(this).closest('div.myButtonContainer')
```

现在选中的是拥有 CSS 类 myButtonContainer 的祖先<div>元素。

其余的方法以类似的方式工作。比如需要找到一个拥有特殊 title 特性的同级按钮：

```
$(this).siblings('button[title="Close"]')
```

这些方法给了我们很大的自由度来根据 DOM 中元素之间的关系选择元素。现在还没有结束。下面接着来看 jQuery 如何更进一步地处理包装集。

2.3.5　更多处理包装集的方式

这一切似乎还不够，jQuery 还有一些技巧来允许我们调整包装对象的集合。

① parents()和 closest()的行为类似，不过有一些细微的差别，比如 closest()从当前元素开始查找，返回的 jQuery 对象可能包含零个或者一个元素；parents()则从父元素开始查找，返回的 jQuery 对象可能包含零个、一个或者多个元素。

find()方法可以将包装集中所有元素的后代全部搜索一遍,并返回包含与传入的选择器表达式相匹配的所有元素的新包装集。例如，给定一个包装集变量 wrappedSet，就可以获取另一个由段落中的所有引用（<cite>元素）组成的包装集，这些段落是原始包装集中的后代元素。

```
wrappedSet.find('p cite')
```

和很多其他的 jQuery 包装器方法一样，当在 jQuery 操作链中调用 find()方法时，它才会发挥真正威力。

方法语法：find

find(selector)

返回新的包装集，它包含原始包装集中与传入的选择器表达式相匹配的元素的所有后代元素

参数

selector （字符串）一个 jQuery 选择器，只有匹配此选择器的元素才能成为返回集合的一部分

返回值

新建的包装集

当我们需要在 jQuery 方法链的中间将搜索范围限制在后代元素中时，这个方法会非常方便，因为我们不可能使用其他的上下文或者约束机制。

本节要探讨的最后一个方法可以对包装集进行检查，看是否包含一个匹配给定选择器的元素。如果至少有一个元素匹配选择器，则 is()返回 true，否则返回 false。例如：

```
var hasImage = $('*').is('img');
```

这个语句会将变量 hasImage 的值设置为 true，前提是当前 DOM 中包含图片元素。

方法语法：is

is(selector)

确定包装集中是否存在与传入的选择器表达式相匹配的元素

参数

selector （字符串）检验包装集中元素的选择器表达式

返回值

如果至少有一个元素与传入的选择器相匹配，则返回 true，否则返回 false

这是 jQuery 中一个高度优化的快速操作，我们可以毫不犹豫地将其用在特别关注性能的领域中。

2.3.6 管理jQuery链

我们非常重视一种能力，即将 jQuery 包装器方法链在一起使用，从而在单个语句中完成多

个任务（我们会继续这么做，因为这意义重大）。这种链式操作的能力不仅允许我们以简洁的方式书写强大的操作代码，而且也能提高效率，因为我们无需重新计算包装集就能向其应用多个方法。

有时候生成的包装集可能会有多个，这取决于方法链中使用的方法。比如，使用 clone()方法（详见第 3 章）生成一个新的包装集，它会创建原始集合的元素副本。一旦生成了新的包装集，就没有办法引用原始集合，从而导致构建多功能的 jQuery 方法链的能力大大减弱。

考虑以下语句：

```
$('img').filter('[title]').hide();
```

这个语句创建了两个包装集：一个是包含了 DOM 中的所有<img>元素的原始包装集，另一个是只包含那些拥有 title 特性元素的包装集。（是的，我们可以使用单个选择器来实现，这里的目的是为了说明后面的概念。想象一下在调用 filter()之前需要在方法链中执行的一些重要操作。）

但是如果我们想随后应用一个方法，比如在对原始包装集进行过滤之后，再添加一个 CSS 类到其中该怎么做呢？我们不能把这个方法添加到现有方法链的尾部，因为这样影响的是拥有 title 特性的图片，而不是原始包装集中的图片。

jQuery 为这种需求提供了 end()方法。当在 jQuery 链中使用这个方法时，它会“回滚”到前一个包装集并将其作为返回值，以便使后续的操作能够应用到前一个包装集。

分析以下代码：

```
$('img').filter('[title]').hide().end().addClass('anImage');
```

filter()方法返回拥有 title 特性的图片集，但是调用 end()方法后会回滚到前一个包装集（包含所有图片的原始集合），该包装集应用了 addClass()方法。如果没有插入 end()方法，addClass()方法就会应用到集合的副本上。

方法语法：end

end()

在 jQuery 方法链中用来将当前的包装集回滚到前一个返回的包装集

参数

无

返回值

前一个包装集

这或许让人联想到，调用 jQuery 方法链时产生的包装集保存在栈中。当调用 end()方法时，最顶层（最近产生的）的包装集从栈中弹出，前一个包装集被公开以便应用后续的方法。

另一个修改包装集栈的便捷方法是 andSelf()，它将栈顶的两个包装集合并为一个包装集。

方法语法：andSelf

andSelf ()

合并方法链中的前两个包装集

参数

无

返回值

合并后的包装集

比如：

```
$('div')
  .addClass('a')
  .find('img')
  .addClass('b')
  .andSelf()
  .addClass('c');
```

这个语句选择所有的<div>元素并向其添加 CSS 类 a，然后创建一个由这些<div>元素后代中所有的<img>元素组成的新包装集，并向这些<img>元素添加 CSS 类 b，最后创建第三个包装集，它是<div>元素和其后代中的<img>元素的并集，并向这些元素添加 CSS 类 c。

哇！最后，<div>元素拥有 CSS 类 a 和 c，而这些元素后代中的<img>元素拥有 CSS 类 b 和 c。

可以看到，jQuery 提供的管理包装集的方法几乎涵盖了需要对包装集进行的任何类型的操作。

2.4 小结

本章重点介绍通过 jQuery 提供的多种方法，识别 HTML 页面中元素，来创建和调整元素集合（在本章以及后面几章中称为包装集）。

jQuery 提供了通用而强大的选择器集合（模仿 CSS 选择器），以便使用简洁而强大的语法来识别页面文档中的元素。这些选择器包含目前大多数现代浏览器都支持的 CSS3 语法。

jQuery 提供的另一个重要功能是，使用 HTML 片段动态地创建新元素，以便创建或扩大包装集。可以将这些孤立的元素连同包装集中的其他元素一起操作，最终添加到页面文档的某些部分中。

jQuery 提供了一组健壮的方法用来调整包装集，以提炼集合的内容，可以在创建之后立即应用这些方法，也可以中途在一组方法链中应用这些方法。对现有包装集应用过滤条件也可以轻松地创建新的包装集。

总之，jQuery 提供了大量工具来确保你可以轻松而精确地找出期望操作的页面元素。

本章中，我们学习了大量的基础知识，但不涉及对页面上的 DOM 元素的具体操作。懂得了如何选择要操作的元素后，就可以开始使用强大的 jQuery DOM 操作方法为页面添加活力了。

第 3 章

用 jQuery 为页面添加活力

本章内容

- 获取和设置元素特性
- 在元素上保存自定义数据
- 操作元素的 CSS 类名
- 设置 DOM 元素的内容
- 存储和获取元素上的自定义数据
- 获取和设置表单元素的值
- 通过添加、移动或者替换元素来修改 DOM 树

还记得过去吗？幸好这些日子已经定格在记忆中——初出茅庐的网页开发者试图为页面添加一些华丽效果却招来反感，比如走马灯、闪烁文本、过去花哨的背景图片（不可避免的影响了页面中文字的可读性）、烦人的 GIF 动画，或许还有更糟糕的，页面加载后不请自播的背景音乐（它们的存在只能测试用户关闭浏览器的速度有多快）。

从那以后，我们走过了漫漫长路。如今，专业的 Web 开发者和设计者都更加理性，使用 DOM 脚本（从前的开发者可能称为动态 HTML 或者 DHTML）所赋予的强大威力来改善用户的 Web 体验，而不是炫耀那些恼人的技巧。

无论是渐进式显示内容、创建除 HTML 基本控件之外的输入控件，还是让用户根据其喜好调整页面，DOM 操作为 Web 开发者提供了很多功能来取悦（而不是惹恼）用户。

几乎每天我们都会打开一些令人惊叹的网页："哇！我不知道网页还可以做那些事情！"。作为称职的专业开发人员（更不用说为了满足对这些事情无限的好奇心），我们立即开始查看源代码，研究他们是**怎么**做到的。

我们无需自己编写所有的脚本，因为 jQuery 提供了一组用来操作 DOM 的健壮工具，完成这些"哇！"页面只需令人惊叹的少量代码。鉴于前一章已经介绍了通过 jQuery 选择 DOM 元素并生成包装集的各种方法，本章将介绍如何利用 jQuery 对这些元素进行操作的强大威力为页面注入活力以及那些令人难忘的"哇！"要素。

3.1 使用元素属性与特性

就 DOM 元素而言，我们可以操作的一些最基本组件就是指定给这些元素的属性和特性。这

些属性和特性最初被指定给 JavaScript 的对象实例，用于呈现解析 HTML 标记生成的 DOM 元素，并且可以在脚本控制下动态地修改。

先来明确一下相关的术语和概念。

属性是 JavaScript 对象的内在性质，每个属性都包含名称和值。JavaScript 的动态特性允许我们在脚本控制下创建 JavaScript 对象的属性。（附录对此概念有详细描述，如果你对 JavaScript 不熟悉可以参考一下。）

特性不是 JavaScript 的原生概念，它只适用于 DOM 元素。特性用于描述 DOM 元素标记中设定的值。

考虑如下图片元素的 HTML 标记：

```
<img id="myImage" src="image.gif" alt="An image" class="someClass"
     title="This is an image"/>
```

在这个元素的标记中，标签名称是 `img`，标记中的 `id`、`src`、`alt`、`class` 和 `title` 描述的是元素的特性，每个特性都由名称和值组成。浏览器会读取并解析这个元素的标记，以便创建在 DOM 中表现此元素的 JavaScript 对象实例。特性被收集到一个列表中，这个列表保存在 DOM 元素实例中的一个名为 `attributes`（非常合理的名字）的属性中。除了在列表中保存特性，每个 DOM 对象还被赋予了多个属性，包括一些描述元素标记中特性的属性。

因此，特性值不仅在 `attributes` 列表中有所体现，还存在于少数属性中。

图 3-1 展示了关于这个过程的简要说明。

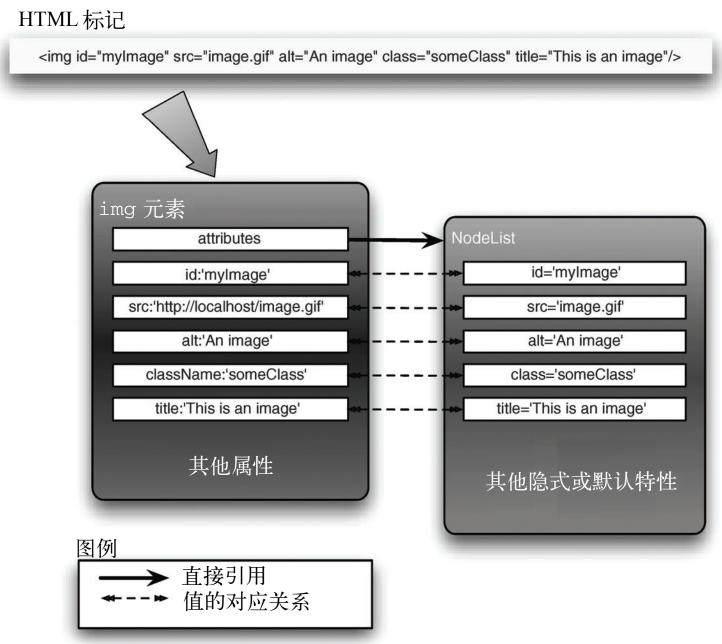


图 3-1　HTML 标记被转化为 DOM 元素，包括标签的特性以及根据标签创建的属性。浏览器为元素的特性和属性之间创建了对应关系

保存在 attributes 列表中的特性值和相应的属性之间有一条活动的连接。改变特性值会导致相应的属性值也发生变化，反之亦然。即便如此，两个值也并不总是完全相同。比如，设置图片元素的 src 特性为 image.gif，会导致 src 属性被设置为图片的绝对 URL。

大部分情况下，JavaScript 属性的名字和相应特性的名字是一致的，但也有一些例外。比如，本例中的 class 特性对应的属性名是 className。

jQuery 不仅提供了操作元素特性的简单方式，而且还允许访问元素实例，从而可以改变其属性。操作特性还是操作属性取决于想做什么以及如何去做。

接下来就从获取和设置元素属性开始。

3.1.1　操作元素属性

jQuery 没有提供获取或修改元素属性的具体方法。不过，我们可以使用原生的 JavaScript 语法来访问属性以及它们的值。关键在于首先获取元素的引用。

不过事实证明，这并不复杂。正如在前一章中看到的，jQuery 提供了很多访问包装集中单个元素的方法。其中的一些方法如下：

- 对包装集使用数组索引，比如$(whatever)[n]；
- 使用 get()方法，通过索引返回单个元素；或者 toArray()方法，返回由集合中的元素组成的数组；
- 使用 each()或 map()方法，在回调函数中可以使用单个元素；
- 使用 eq()方法或者:eq()过滤器；
- 通过某些方法（比如 not()和 filter()）的回调函数，这些方法将元素设置为回调函数的上下文。

以 each()方法为例，可以使用如下代码设置 DOM 中每个元素的 id 属性为元素标签名与元素在 DOM 中位置的组合：

```
$('*').each(function(n){
  this.id = this.tagName + n;
});
```

本例中，我们获取作为回调函数上下文（this）的元素引用，并直接向其 id 属性赋值。

在 JavaScript 中处理特性不像处理属性那么简单明了，因此 jQuery 提供了多种辅助手段来处理它们。下面来看看如何处理特性。

3.1.2　获取特性值

和很多其他 jQuery 方法一样，attr()方法可以用来执行读操作或者写操作。当 jQuery 方法可以进行这两种操作时，传入方法的参数个数和类型就决定了该执行方法的哪个变体。

作为一个具有这种两面性的方法，attr()方法可以用来获取匹配集合中第一个元素的某个特性值，也可以用来设置所有匹配元素的特性值。

执行读操作的 attr()方法变体的语法如下：

方法语法：attr

attr(name)

获取匹配集合中第一个元素指定特性的值

参数

name （字符串）需要获取值的特性名称

返回值

第一个匹配元素的特性值。如果匹配集为空，或者第一个元素不存在此特性，则返回 undefined

虽然我们通常认为元素的特性是由 HTML 预定义的，但是也可以使用 attr()，通过 JavaScript 或者 HTML 标记来设置自定义特性。为了说明这一点，我们修改前一个示例中的<img>元素，为其添加一个自定义标记特性（粗体）：

```
<img id="myImage" src="image.gif" alt="An image" class="someClass"
     title="This is an image" data-custom ="some value"/>
```

注意，我们已经为这个元素添加了一个自定义特性，并取了一个缺乏想象力的名称 data-custom。可以使用以下代码获取特性的值，就好像它是一个标准特性：

```
$("#myImage").attr("data-custom")
```

自定义特性与 HTML

在 HTML 4 中，使用非标准的特性名称（比如 data-custom）虽然是一个常用的技巧，但是却会导致标记是无效的，并且不会通过官方的验证测试[①]。如果你比较关注验证的结果，那就要谨慎行事了。

另一方面，HTML 5 正式承认并允许这样的自定义特性，只要自定义特性的名称以字符串 data-为前缀即可。根据 HTML 5 的规则，任何遵循此命名约定的特性都是有效的；而不遵循此约定的还是无效的。（详情请参考 W3C 的 HTML 5 规范定义：http://www.w3.org/TR/html5/dom.html#attr-data。）

预料到 HTML 5 的流行，我们在示例中采用 data-为前缀的命名规则。

在 HTML 中特性的名称是不区分大小写的。无论在标记中如何声明特性（例如 title），都可以使用任意大小写变体（Title、TITLE、TiTlE）来访问（或设置，下一节会介绍）特性，而且其他等价组合也可以完成。在 XHTML 中，尽管在标记中特性的名称必须是小写的，不过也可以使用任意大小写变体获取特性值。

此时你可能会问，“既然访问属性这么简单（参见 3.1.1 节），为什么还要处理特性呢？”

问题的答案是，jQuery 的 attr()方法不仅仅是 JavaScript 的 getAttribute()和 setAttribute()方法的包装器。除了允许对元素特性集的访问之外，jQuery 还允许访问一些常用属性，而传统上这一操作对页面开发者而言是一件痛苦的事情，因为它处处依赖于浏览器的实现。

① 这里指的是 W3C 标记验证服务：http://validator.w3.org/。

表 3-1 展示了规范化的访问名称的集合。

表3-1　jQuery的**attr()**方法的规范化访问名称

jQuery规范化名称	DOM名称
`cellspacing`	`cellSpacing`
`class`	`className`
`colspan`	`colSpan`
`cssFloat`	IE下是`styleFloat`，其他浏览器下是`cssFloat`
`float`	IE下是`styleFloat`，其他浏览器下是`cssFloat`
`for`	`htmlFor`
`frameborder`	`frameBorder`
`maxlength`	`maxLength`
`readonly`	`readOnly`
`rowspan`	`rowSpan`
`styleFloat`	IE下是`styleFloat`，其他浏览器下是`cssFloat`
`tabindex`	`tabIndex`
`usemap`	`useMap`

除了这些有用的快捷方式外，`attr()`的设置操作变体还有一些便捷的功能。下面就来看一下。

3.1.3　设置特性值

使用 jQuery 来设置包装集中元素的特性有两种方法。先从最直接的方法开始，这个方法允许一次设置一个特性（对包装集中的所有元素），语法如下所示。

方法语法：`attr`

`attr(name, value)`

将包装集中所有元素的已命名的特性设置为传入的值

参数

`name`　（字符串）要设置的特性名称

`value`　（任意|函数）指定特性的值。既可以是生成一个值的任意 JavaScript 表达式，也可以是一个函数。为每个包装元素分别调用此函数，传递元素的索引和已命名特性的当前值。函数的返回值会成为新的特性值

返回值

包装集

`attr()`方法的这个变体初看起来很简单，其实它的执行过程是相当复杂的。

在这个变体最基本的形式中，当 `value` 参数是可以生成值的任意 JavaScript 表达式（包括数组）时，设置表达式的计算结果为特性值。

当 `value` 参数是一个函数引用时，事情变得更加有趣。这种情况下，会为包装集中每个元素分别调用函数，函数的返回值会作为特性值。当函数被调用时，需要向其传入两个参数：一个

是元素在包装集中从零开始的下标，另一个是已命名特性的当前值。此外，在函数调用中，将当前元素设置为函数的上下文（this），允许函数为每个具体元素调整其处理过程——这也是以这种方式使用函数的强大之处。

考虑如下语句：

```
$('*').attr('title',function(index,previousValue) {
  return previousValue + ' I am element ' + index +
         ' and my name is ' + (this.id || 'unset');
});
```

这个方法会作用于页面上的所有元素，通过为每个元素的 title 特性附加一个字符串（由元素在 DOM 中的下标值和每个元素的 id 特性值组成）来修改元素的 title 特性。

我们会使用这种方式来指定特性值但前提是元素的特性值依赖于元素其他特性值或需要根据原来的值来计算新的值，或者由于其他一些原因需要单独设置每个值。

attr()方法第二种形式的变体允许通过一条语句一次设置多个特性值。

方法语法：attr

attr(attributes)

用传入的对象指定的属性和值来设置匹配集中所有元素相应的特性值

参数

attributes　（对象）一个对象，其属性会复制到包装集中所有元素的特性值上

返回值

包装集

这是一种快速而简单的方式，用来设置包装集中所有元素的多个特性值。传入的参数可以是任意的对象引用，通常是对象的字面值，其属性指定了要设置的特性的名称和值。考虑下面这段代码：

```
$('input').attr(
  { value: '', title: 'Please enter a value' }
);
```

这个语句设置所有<input>元素的值为空字符串，并将 title 设置为字符串 Please enter a value。

注意，如果作为 value 参数传入的对象的属性值是一个函数引用，则它的操作方式类似于前面介绍的 attr()方法，并且会为匹配集中的每个元素分别调用函数。

警告　IE 浏览器不允许修改<input>元素的 name 或 type 特性。如果想在 IE 中修改<input>元素的名称或类型，就必须使用拥有指定名称或类型的新元素来替换此元素。这同样适用于 file 的 value 特性或 password 类型的<input>元素。

现在我们已经知道了如何获取和设置特性了，那么如何删除它们呢？

3.1.4　删除特性

为了从 DOM 元素中删除特性，jQuery 提供了 `removeAttr()`方法。它的语法如下所示。

方法语法：removeAttr

removeAttr (name)

从每个匹配元素中删除指定的特性

参数

name	（字符串）要删除的特性名称

返回值

包装集

注意，删除一个特性不会删除 JavaScript DOM 元素中相应的属性，尽管这可能会导致属性值的改变。比如，从元素中删除 `readonly` 特性会导致元素的 `readOnly` 属性值由 `true` 变为 `false`，但是属性本身不会从元素中删除。

现在就通过一些示例来学习如何在页面中运用这些知识。

3.1.5　有趣的特性

看看如何使用这些方法来以不同的方式处理元素的特性。

1. 实例#1——强制在新窗口中打开链接

假设我们想要在新窗口中打开站点上所有指向外部域名的链接。如果能完全控制整个标记并能为它们添加 `target` 特性，那这实现起来就相当简单，如下所示：

```
<a href="http://external.com" target="_blank">Some External Site</a>
```

这样确实不错，但是如果不能控制标记该怎么办？我们或许在运行一个内容管理系统或者一个维基站点，在这些站点上最终用户能够添加内容，但不可能指望用户为所有的外部链接都加上 `target="_blank"`。首先要明确目标：我们想要在新窗口中打开所有 `href` 特性以 `http://`开头的链接（我们已经确定这可以通过设置 `target` 特性值为`_blank` 来实现）。

可以使用在本节学到的技巧简洁地实现，如下所示：

```
$("a[href^='http://']").attr("target","_blank");
```

首先选择 `href` 特性以 `http://`开头的所有链接（这表明引用是外部的），然后设置它们的 `target` 特性值为`_blank`。完成这个任务只需要一行 jQuery 代码。

2. 示例#2——解决可恶的双重提交问题

jQuery 特性功能的另一个极佳用途是帮助解决一个长期存在于 Web 应用（富因特网应用等）中的问题：可恶的双重提交问题。这是 Web 应用的常见难题，当表单提交时会出现延迟，有时是几秒或者更长，这使得用户会多次单击提交按钮，从而给服务器端代码带来了诸多麻烦。

作为客户端的解决方案（服务器端代码仍然需要保持警惕），我们捕获表单的 submit 事件并在第一次单击后禁用提交按钮。这样，用户将没有机会多次单击提交按钮，而且会看到可见的指示（假设已禁用的按钮在浏览器中这样表现），用来提示表单正在提交处理中。不用为下面示例中有关事件的处理细节担心（详见第 4 章），目前只需要专注于 attr()方法的使用即可：

```
$("form").submit(function() {
  $(":submit",this).attr("disabled", "disabled");
});
```

在事件处理器的主体部分，我们通过:submit 选择器获取表单中的所有提交按钮，并且将其 disabled 特性设置为"disabled"（W3C 官方推荐设置的特性值）。注意，当创建匹配集时，我们提供了上下文值 this（第二个参数）。第 4 章研究事件处理时你会发现，在事件处理器内操作时，this 指针总是指向触发此事件的页面元素。在本例中，this 指向的就是表单实例。

3

什么时候"enabled"的元素没有被禁用

不要想当然地认为可以用下面的语句将 disabled 的值替换为 enabled：

```
$(whatever).attr("disabled","enabled");
```

并且期望此元素从此变得可用了。这句代码仍然会禁用此元素！

根据 W3C 的规则，正是 disabled 特性的存在（而不是特性的值）才使得此元素被设置为禁用状态。因此它的值是什么根本不重要。如果 disabled 特性存在，此元素就被禁用了。

因此，为了重新启用此元素，可以删除此特性，也可以使用 jQuery 提供的简单方式。如果为此特性值提供布尔类型值 true 或 false（不是字符串"true"或者"false"），jQuery 将在后台完成正确的事情，false 则删除此特性，true 则添加此特性。

警告 使用这种方式来禁用提交按钮（或多个按钮）不能免除服务器端代码的职责：防止双重提交或执行任何其他类型的验证。为客户端代码添加这种功能可以提高页面对最终用户的友好度，而且能帮助阻止正常情况下的双重提交问题。但是这不能阻止攻击或者其他黑客行为，因此服务器端代码必须继续保持警惕。

对 HTML 和 W3C 定义的数据而言，元素的特性和属性是非常有用的概念，但是在页面创作过程中，我们经常需要存储自定义数据。下面来看看在这方面 jQuery 可以为我们做什么。

3.1.6 在元素上存储自定义数据

我们要直截了当地表明态度：全局变量真糟糕。

除了极少数真正必要的情况外，在定义和执行复杂的页面行为时，很难想象得出有比全局变量更糟糕的信息存储方式了。这不仅因为会遇到作用域的问题，而且在多个操作并发执行时（打开和关闭菜单、触发 Ajax 请求、执行动画，等等），其扩展性也不好。

JavaScript 的原生函数可以帮助我们使用闭包解决部分问题，但是闭包也不能彻底解决问题，而且也不适用于所有情况。

因为页面行为是面向元素的，所以使用元素本身作为存储范围是合理的。再者，JavaScript 的本质及其动态创建对象自定义属性的能力也可以帮助我们解决问题。不过我们必须谨慎行事。DOM 元素以 JavaScript 对象实例的形式存在，就像所有其他的对象实例一样，这样就可以使用我们自己选择的自定义属性来扩展它们。但这却暗藏危机！

这些自定义属性（也就是所谓的 expandos）是有风险的。特别是它可以很容易地创建循环引用，从而导致严重的内存泄漏。在传统的 Web 开发中，当新的页面加载后老的 DOM 时常被丢弃了，因此内存泄漏可能不会成为大问题。但是对于我们来说，作为高交互性 Web 应用程序的开发者，在页面中引入大量长时间运行的脚本，内存泄漏就有可能成为一个严重的问题。

jQuery 提供了一种解决方案，以一种可控的方式将数据附加到所选择的 DOM 元素上，这样就不需要依靠有潜在问题的 expandos 了。我们可以将任意的 JavaScript 值、甚至数组和对象通过名为 `data()`的巧妙方法附加到 DOM 元素上。语法如下所示。

方法语法：`data`

`data (name, value)`

将传入的值添加到为所有包装元素准备的 jQuery 托管数据仓库中

参数

`name`	（字符串）要存储的数据名称
`value`	（对象|函数）要存储的值。如果值是个函数，则为每个包装元素调用此函数，传递当前元素作为函数的上下文。函数的返回值作为数据值

返回值

包装集

只写的数据不是特别有用，因此必须要有一个获取已命名数据的方法。再次使用 `data()`方法是顺理成章的事情。使用 `data()`方法获取数据的语法如下所示。

方法语法：`data`

`data (name)`

通过指定的名称来获取之前在包装集的第一个元素上保存的任何数据

参数

`name`	（字符串）要获取的数据名称

返回值

获取的数据，如果没有找到则返回 `undefined`

考虑到内存管理的需要，jQuery 也提供了 `removeData()`方法来丢弃那些不再需要的数据。

方法语法：removeData

removeData (name)
通过指定的名称来删除之前在包装集的所有元素上保存的数据
参数
name （字符串）要删除的数据名称
返回值
包装集

注意，在使用 jQuery 方法从 DOM 中删除一个元素时不需要“手工”删除数据。jQuery 将会很聪明地帮我们处理这个事情。

在接下来几章的很多示例中你将看到，将数据附加到 DOM 元素上的能力对我们来说如虎添翼，对于被全局变量搞的晕头转向的开发者来说，将数据存储在元素层次结构的上下文中为应用打开了一个全新的世界。本质上，DOM 树已经成为了一个完整“命名空间”层次结构，我们不再局限于单个的全局空间。

在本节的前面部分我们曾提到 `className` 属性，作为说明标记特性名称不同于属性名称的示例。但事实上，类名在其他方面也有点特别，因此 jQuery 对其进行了特殊处理。下一节将会阐述比直接访问 `className` 属性或使用 `attr()`方法更好的处理类名的方式。

3.2 改变元素样式

改变元素样式的方法有两种。可以添加或删除一个类名，使得现有的样式表基于新的类来重新设置元素的样式。也可以操作 DOM 元素，直接为其应用样式。

下面来看下 jQuery 如何通过类来修改元素样式，从而简化样式的修改方式。

3.2.1 添加和删除类名

DOM 元素的 `class` 特性的格式和语义与众不同，并且对于创建交互式界面至关重要。为元素添加类名或从元素中删除类名，是动态修改元素渲染样式的主要手段。

可以为每个元素指定任意数量的类名，这是导致元素类名非常独特的一方面原因（也是需要应对的一个挑战）。在 HTML 中，`class` 特性用来提供以空格分隔，由多个类名所组成的字符串。例如：

```
<div class="someClass anotherClass yetAnotherClass"></div>
```

遗憾的是，DOM 元素相应的 `className` 属性不是以类名组成的数组，而是以空格分隔的字符串。多麻烦啊！这太令人失望了。这就意味着每当我们向一个拥有类名的元素中添加（或删除）

类名时，就需要在读字符串的时候对其进行解析以确定单个名称，并且在写的时候确保恢复成有效的空格分隔的格式。

注意　类名列表是无序的。也就是说，以空格分隔的列表中类名的顺序没有语义上的意义。

虽然编写代码来处理所有这些事情不是艰巨的任务，不过将操作的细节隐藏起来并抽象出 API 总是个好的想法。幸运的是，jQuery 已经为我们完成了这一切。

为匹配集中所有的元素添加类名，利用 `addclass()`方法就很简便。

方法语法：addClass

`addClass (names)`

为包装集中的所有元素添加指定的单个或多个类名

参数

`names`	（字符串|函数）指定要添加的单个类名，或者以空格分隔的字符串表示的多个类名。如果参数是一个函数，则为每个包装元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素的下标和当前类的值。函数的返回值作为单个或者多个类名

返回值

包装集

删除类名可以直接使用 `removeClass()`方法来实现。

方法语法：removeClass

`removeClass (names)`

从包装集中的每个元素上删除指定的单个或多个类名

参数

`names`	（字符串|函数）指定要删除的单个类名，或者以空格分隔的字符串表示的多个类名。如果参数是一个函数，则为每个包装元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素的下标和删除操作之前的类名的值。函数的返回值作为将要删除的单个或者多个类名

返回值

包装集

我们可能经常想要来回切换一组样式，这可能是为了表示两个状态之间的变化，也可能是出于界面上有意义的任何其他原因。jQuery 通过 `toggleClass()`方法简单化操作。

方法语法：toggleClass

toggleClass (names)

如果元素不存在指定类名则为其添加此类名，如果元素已经拥有这个类名则从中删除此类名。注意，会分别检测每个元素，因此一些元素可能被添加了类名，而另外一些则可能被删除了类名

参数

names	（字符串|函数）指定要切换的单个类名，或者以空格分隔的字符串表示的多个类名。如果参数是一个函数，则为每个包装元素调用此函数，设置当前元素为函数上下文。函数的返回值作为单个或者多个类名

返回值

包装集

当我们需要快速简洁地在元素之间切换显示效果时，`toggleClass()`方法就能派上用场了。考虑一个“斑马条纹”示例，我们希望为表格中的行交替设置不同的颜色。假设出于某些合理的理由，我们需要在某些事件发生时，将奇数行的彩色背景与偶数行的背景进行互换（还可能再换回来）。`toggleClass()`方法可以轻松实现在为隔行添加类名的同时，从剩余的行中删除类名。

下面就来试一试。在文件 chapter3/zebra.stripes.html 中，你可以找到一个包含车辆信息表格的页面。在页面的脚本定义中，定义了如下的一个函数：

```
function swapThem() {
  $('tr').toggleClass('striped');
}
```

这个函数使用 `toggleClass()`方法为所有`<tr>`元素切换名为 `striped` 的类。我们也定义了文档就绪处理器，如下所示：

```
$(function(){/

  $("table tr:nth-child(even)").addClass("striped");
  $("table").mouseover(swapThem).mouseout(swapThem);
});
```

该处理器主体中的第一个语句使用了我们在前一章中学过的 `nth-child` 选择器，为表格中的隔行元素应用类 `striped`。第二个语句为 `mouseover` 和 `mouseout` 事件创建了两次都调用同一个函数 `swapThem` 的事件处理器。我们会在下一章中全面学习事件处理，不过现在的重点是当鼠标进入或者移出表格时，会执行以下代码：

```
$('tr').toggleClass('striped');
```

结果是每当鼠标光标进入或者移出表格时，就会删除所有拥有 `striped` 类的`<tr>`元素，并向所有没有 `striped` 类的`<tr>`元素添加此类。这个（有点烦人的）行为在图 3-2 中分两部分显示。

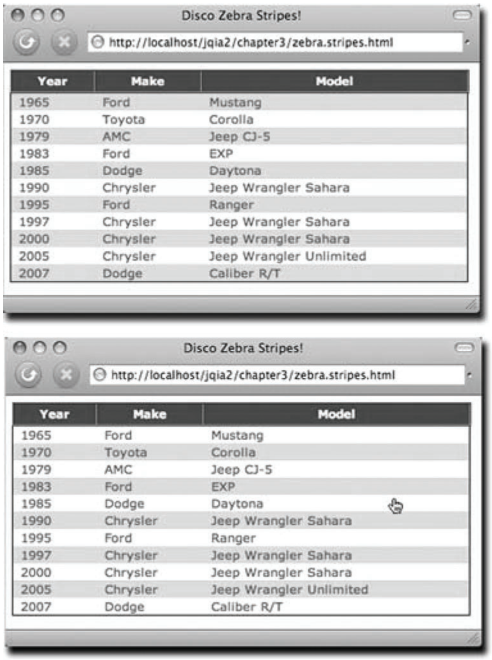


图 3-2　每当鼠标光标进入或者移出表格时，切换 `striped` 类的有无状态

根据元素是否已经拥有类来进行类切换是十分常见的一种操作，不过根据其他任意条件来切换类也很常见。对于这种更普遍的情况，jQuery 提供了 `toggleClass()`方法的另一个变体，它可以让我们根据任意的布尔表达式来添加或者删除类。

方法语法：`toggleClass`

`toggleClass (names, switch)`

如果 `switch` 表达式的计算结果为 `true`，则添加此类名；如果计算结果为 `false`，则删除此类名

参数

`names`	（字符串|函数）指定要切换的单个类名，或者以空格分隔的字符串表示的多个类名。如果参数是一个函数，则为每个包装元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素的下标和当前类的值。函数的返回值作为单个或者多个类名
`switch`	（布尔）一个控制表达式，它的值决定了是要为元素添加类（值为 `true`）还是删除类（值为 `false`）

返回值

包装集

判断元素是否包含特殊的类是很常见的需求。例如，可能需要根据元素是否含有特定类来进行条件转移，或者只是想使用类来识别某一类型的元素。

使用 jQuery，就可以通过调用 hasClass()方法来实现这些功能：

```
$("p:first").hasClass("surpriseMe")
```

如果匹配集中有包含指定类的元素，则此方法将返回 true。该方法语法如下所示。

方法语法：hasClass

hasClass (name)

确定匹配集中是否有元素拥有通过 name 参数传入的类名

参数

name　　（字符串）要检查的类名

返回值

如果包装集中有元素拥有传入的类名，则返回 true；否则返回 false

回想下第 2 章中的 is()方法，以下代码也可以实现相同的效果：

```
$("p:first").is(".surpriseMe")
```

但是可以证明，hasClass()方法有助于编写可读性更高的代码。从内部实现来看，hasClass()也更为高效。

另一个常见需求是，以数组的形式返回特定元素的类名列表，而不是难以处理的以空格分隔的列表。可以尝试编写以下代码来实现：

```
$("p:first").attr("className").split(" ");
```

我们介绍过，如果查询的特性不存在，attr()方法将会返回 undefined，因此如果<p>元素没有任何类名这个语句就会抛出错误。

可以通过先检查特性是否存在来解决这个问题，如果想将整个代码包装在一个可重用的、有效 jQuery 扩展中，可以这样写：

```
$.fn.getClassNames = function() {
  var name = this.attr("className");
  if (name != null) {
    return name.split(" ");
  }
  else {
    return [];
  }
};
```

不必担心用来扩展 jQuery 的语法细节，第 7 章将详细探讨这个主题。重要的是一旦定义了这个扩展，就可以在脚本的任何地方使用 getClassNames()方法来获取由类名组成的数组，如果元素没有任何类，此方法将返回空数组。太棒了！

CSS 类名是操作元素展现样式的一个强大工具，但有时我们想直接操作样式本身的细节，比如直接在元素上声明样式。来看看 jQuery 为此提供了哪些功能。

3.2.2　获取和设置样式

尽管修改元素的类特性允许我们选择要应用的一组预定义样式表规则，但有时候我们想要整体覆盖样式表。直接将样式应用到元素上（通过可以从所有 DOM 元素上获取的 style 属性）将会自动覆盖样式表，这让我们可以更精确地控制单个元素及其样式。

jQuery 的 css()方法允许我们操作这些样式，其工作方式类似于 attr()方法。可以通过指定名称和值来设置单个的 CSS 样式，也可以通过传入一个对象来设置一系列 CSS 样式。首先来看看设置单个样式的名称和值。

方法语法：css

css(name, value)

设置每个匹配元素的已命名 CSS 样式属性为指定的值

参数

name	（字符串）要设置的 CSS 属性名称
value	（字符串|数字|函数）一个包含属性值的字符串、数字，或函数。如果传入的参数是函数，则为包装集中的每个元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素下标和 CSS 属性的当前值。函数的返回值作为 CSS 属性值的新值

返回值

包装集

如上所述，参数 value 接受函数的方式类似于 attr()方法。这意味着（例如），可以将包装集中所有元素的宽度扩大 20 像素，如下所示：

```
$("div.expandable").css("width",function(index, currentWidth) {
   return currentWidth + 20;
});
```

还有一点值得注意，（也是 jQuery 为开发提供的另一种便利。）通过传递一个 0.0~1.0 之间的值，就可以让通常有问题的 opacity 属性完美地跨浏览器工作，不再需要组合使用 IE alapha 滤镜、-moz-opacity 以及其他类似的值！

下面来看看 css()方法的快捷形式，它的工作方式和 attr()的快捷版本完全一致。

方法语法：css

css(properties)

设置所有匹配元素的 CSS 属性为传入对象的多个键值

参数

properties　（对象）指定一个对象，其属性被复制为包装集中所有元素的 CSS 属性

返回值

包装集

在前一章的表 2-1 中，我们已经看到这个方法的变体非常有用。为了减少你翻页的麻烦，我们把相关的代码再次列出来：

```
$('<img>',
  {
    src: 'images/little.bear.png',
    alt: 'Little Bear',
    title:'I woof in your general direction',
    click: function(){
      alert($(this).attr('title'));
    }
  })
  .css({
    cursor: 'pointer',
    border: '1px solid black',
    padding: '12px 12px 20px 12px',
    backgroundColor: 'white'
  })
  ...
```

和在 attr()方法的快捷版本里一样，在 properties 参数对象中我们可以使用函数作为任何 CSS 属性的值，包装集中的每个元素会分别调用该函数，以确定将要应用的值。

最后，可以通过将名称传入 css()方法，以便获取已计算的与此名称关联的属性样式。当提到已计算的样式时，指的是应用了所有链接、嵌入和内联 CSS 之后的样式。令人印象深刻的是，这可以在所有浏览器下完美地工作，甚至对 opacity 也一样，它会返回表示一个字符串，表示 0.0~1.0 之间的数字。

方法语法：css

css(name)

获取包装集中第一个元素的 CSS 属性的已计算值，这个 CSS 属性由 name 指定

参数

name　（字符串）指定一个 CSS 属性的名称，返回它的已计算样式值

返回值

字符串形式的已计算样式值

切记css()方法的这个变体总是返回字符串，因此如果需要数字或其他的类型时，就必须解析这个返回的字符串。

这并不总是很方便，因此对于小部分经常访问的 CSS 值，jQuery 体贴地提供了很便捷的方法来访问这些值并将其转化为最常用的类型。

1. 获取和设置尺寸

如果想在页面上设置或获取 CSS 样式时，元素的宽度和高度应该是最常用的一组属性。jQuery 允许我们以数字值而不是字符串的方式来处理元素的尺寸，这让事情变得简单很多。

具体而言，我们可以使用便捷的width()和height()方法来获取（或设置）元素的宽度和高度的数值。可以这样来设置元素的宽度和高度，如下所示。

方法语法：width 和 height

width(value) height(value)

设置匹配集中所有元素的宽度和高度

参数

value	（数值|字符串|函数）要设置的值。它可以是以像素为单位的数字，也可以是以单位（比如 px、em、%）表示的字符串值。如果没有指定单位，则默认为 px。 如果参数是一个函数，则为每一个包装元素调用此函数，传递当前元素作为函数上下文。函数的返回值作为要设置的值

返回值

包装集

切记这两个方法是更为详细的css()函数快捷方式，因此

```
$("div.myElements").width(500)
```

等价于

```
$("div.myElements").css("width",500)
```

可以像下面这样来获取宽度和高度。

方法语法：width 和 height

width()

height()

获取包装集中第一个元素的宽度或高度

参数

无

返回值

已计算的宽度或高度，它是以像素为单位的数值

事实上，以数值形式返回宽度和高度值并不是这些函数提供的唯一便利。如果你尝试通过查看元素的 `style.width` 和 `style.height` 属性来获取其宽度和高度，就会面对令人沮丧的事实：只能通过元素相应的 `style` 属性来设置这些属性。为了通过这些属性来获取元素的尺寸，必须首先设置它们。这并不是很有用！

另一方面，`width()`和 `height()`方法计算并返回元素的大小。在流式布局的简单页面，知道元素的精确尺寸通常不是很必要，但是在高交互的脚本页面，知道这些精确的尺寸对于正确地放置活动元素（比如上下文菜单、自定义工具提示、已扩展控件以及其他动态组件）至关重要。

现在让这两个方法工作起来吧。图 3-3 所示的示例页面由两个基本元素组成：一个是作为测试对象的`<div>`元素，它包含一段文本（以边框和背景颜色突出显示），另一个是用来显示测试对象尺寸的`<div>`。

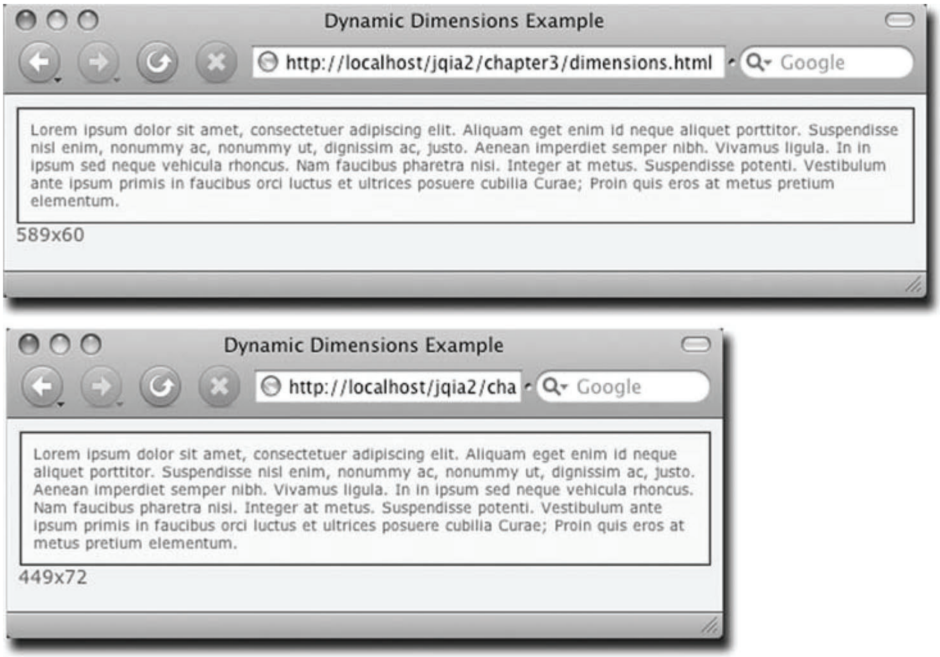


图 3-3　测试元素的宽度和高度是不固定的，它们取决于浏览器窗口的宽度

因为没有应用指定尺寸的样式规则，所以测试对象的尺寸事先并不知道。元素的宽度是由浏览器窗口的宽度决定的，而它的高度取决于需要多少空间来显示包含的文本。改变浏览器窗口的大小会导致元素的高度和宽度发生改变。

我们在页面中定义了一个函数，这个函数将会使用 `width()`和 `height()`方法获取测试对象`<div>`（识别为 `testSubject`）的尺寸，并将结果值显示在第二个`<div>`（识别为 `display`）中。

```
function displayDimensions() {
  $('#display').html(
    $('#testSubject').width()+'x'+$('#testSubject').height()
  );
```

在页面的就绪处理器中调用这个函数。结果显示，在那个特定的浏览器窗口中 display 元素的初始值显示为 589 像素×60 像素，如图 3-3 的上半部分所示。

在窗口的尺寸调整处理器中也调用了同一个函数，它会在浏览器窗口改变时更新显示结果，如图 3-3 下半部分所示。

随时确定元素的已计算尺寸，这个能力对于精确定位页面上的动态元素至关重要。

这个页面的完整代码如代码清单 3-1 中所示，也可以在文件 chapter3/dimensions.html 中找到。

代码清单 3-1　动态追踪并显示元素的尺寸

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Dimensions Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <style type="text/css">
      body {
        background-color: #eeeeee;
      }
      #testSubject {
        background-color: #ffffcc;
        border: 2px ridge maroon;
        padding: 8px;
        font-size: .85em;
      }
    </style>
    <script type="text/javascript"
            src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript">
      $(function(){
        $(window).resize(displayDimensions);
        displayDimensions();
      });

      function displayDimensions() {
        $('#display').html(
          $('#testSubject').width()+'x'+$('#testSubject').height()
        );
      }
    </script>
  </head>

  <body>
    <div id="testSubject">
      Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
      Aliquam eget enim id neque aliquet porttitor. Suspendisse
      nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
      Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
      sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
```

建立调用 **report()** 函数的尺寸调整处理器

在文档就绪处理器中调用 **report()** 函数

显示测试对象的宽度和高度

使用模拟文本声明测试对象

```
        Integer at metus. Suspendisse potenti. Vestibulum ante
        ipsum primis in faucibus orci luctus et ultrices posuere
        cubilia Curae; Proin quis eros at metus pretium elementum.
      </div>
      <div id="display"></div>          <-- 在此区域显示宽度和高度
    </body>
  </html>
```

除了非常方便的 width()和 height()方法，jQuery 还提供了类似的方法用来获取更加特殊的尺寸值，表 3-2 列出了这些方法。

表3-2 其他与尺寸相关的jQuery方法

方 法	描 述
innerHeight()	返回第一个匹配元素的“内部高度”，不包含边框但包含内边距
innerWidth()	返回第一个匹配元素的“内部宽度”，不包含边框但包含内边距
outerHeight(margin)	返回第一个匹配元素的“外部高度”，包含边框和内边距。如果margin参数为true，则包含外边距，否则忽略外边距
outerWidth(margin)	返回第一个匹配元素的“外部宽度”，包含边框和内边距。如果margin参数为true，则包含外边距，否则忽略外边距

当处理窗口或文档元素时，推荐使用 width()和 height()，避免使用内部或者外部方法。[①]

还没有结束，jQuery 还提供了轻松支持定位和滚动值的方法。

2. 定位和滚动

jQuery 提供了两个获取元素位置的方法。这两个方法都返回一个 JavaScript 对象，这个对象包含两个属性：top 和 left，很明显这两个属性分别用来表示元素上边框和左边框的值。

这两个方法使用不同的参照源来测量元素相对于参照源的计算值。其中的一个方法 offset()，返回相对于文档的位置。

方法语法：offset

offset()

返回包装集中第一个元素相对于文档参照源的位置（以 px 为单位）

参数

无

返回值

一个用浮点数（通常四舍五入为最近的整数）来表示 left 和 top 属性的 JavaScript 对象，用来描述相对于文档源的以像素为单位的位置

另一个方法是 position()，返回以元素的最近偏移父元素为参照源的相对位置。

① $(window).innerHeight()会抛出异常，而$(window).outerHeight()则返回 NaN。

方法语法：position

position()

返回包装集中第一个元素相对于最近偏移父元素的位置（以 px 为单位）

参数

无

返回值

一个以整数来表示 `left` 和 `top` 属性的 JavaScript 对象，用来描述相对于最近偏移父元素的位置（以 px 为单位）

元素的偏移父元素是拥有显式定位规则 `relative` 或 `absolute` 的最近的祖先元素。

`offset()`和`position()`都只能对可见元素使用，我们推荐所有的内边距、边框、外边距都使用像素值以便获取精确的结果。

除了元素定位，jQuery 还赋予了我们获取和设置元素滚动位置的能力。表 3-3 描述了这些方法。

表 3-3 中的所有方法对可见和隐藏元素都有效。

表3-3 jQuery滚动控制方法

方 法	描 述
`scrollLeft()`	返回第一个匹配元素的水平滚动偏移值
`scrollLeft(value)`	设置所有匹配元素的水平滚动偏移值
`scrollTop()`	返回第一个匹配元素的垂直滚动偏移值
`scrollTop(value)`	设置所有匹配元素的垂直滚动偏移值

既然已经学了如何获取和设置元素的样式，接下来就开始学习修改其内容的各种方法。

3.3 设置元素内容

当谈到修改元素的内容时，有一场正在进行的关于哪种技术更好的争论：是使用 DOM API 方法还是改变元素的内部 HTML。

尽管使用 DOM API 方法肯定是正确的，但是它相当“啰嗦”并且会产生大量代码，其中的大部分代码很难用肉眼来检查其正确性。在大多数情况下，修改元素的 HTML 更加简单、高效，因此 jQuery 提供了很多方法来完成这一任务。

3.3.1 替换HTML或者文本内容

首先是简单的`html()`方法，当不带使用参数调用时，就获取元素的 HTML 内容；当带参数调用时，就像其他 jQuery 函数那样，设置元素的 HTML 内容。

下面是获取元素的 HTML 内容的语法。

方法语法：html

html()

获取匹配集中第一个元素的 HTML 内容

参数

无

返回值

第一个匹配元素的 HTML 内容。返回的值与所访问元素 innerHTML 属性是一样的

下面是设置所有匹配元素的 HTML 内容的语法。

方法语法：html

html(content)

将传入的 HTML 片段设置为所有匹配元素的内容

参数

content	（字符串|函数）要设置为元素内容的 HTML 片段。如果参数是一个函数，则会为每一个包装元素调用此函数，设置当前元素为函数上下文，并传递两个参数：元素的下标和当前元素的内容。函数的返回值作为新的内容

返回值

包装集

也可以只设置或者获取元素的文本内容。不带参数调用 text()方法，就会返回所有文本组合而成的字符串。例如，假设有如下 HTML 片段：

```
<ul id="theList">
  <li>One</li>
  <li>Two</li>
  <li>Three</li>
  <li>Four</li>
</ul>
```

如下语句：

```
var text = $('#theList').text();
```

把变量 text 设置为 OneTwoThreeFour。

方法语法：`text`

`text()`

将包装元素中的所有文本内容连接起来，并作为此方法的结果返回

参数

无

返回值

连接而成的字符串

也可以使用 `text()` 方法来设置包装元素的文本内容。这个格式的语法如下所示。

方法语法：`text`

`text(content)`

设置所有包装元素的文本内容为传入的值。如果传入的文本包含尖括号（<和>）或 AND 符号（&），则这些字符会被替换为其相应的 HTML 实体字符[1]

参数

`content`　（字符串|函数）要设置到包装元素中的文本内容。所有的尖括号字符都会被转义为 HTML 实体字符。如果参数是一个函数，则为每个包装元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素的下标和元素现有的文本内容。函数的返回值作为新的内容

返回值

包装集

注意，使用这些方法设置元素的内部 HTML 或文本，将会替换掉之前在元素中的内容，因此使用这些方法必须小心。如果不想覆盖元素原先的全部内容，还有很多其他的方法可以保持元素原本的内容不变，但是会修改它们的内容或者其周围的元素[2]。下面来看看这些方法。

3.3.2　移动和复制元素

无需重新加载页面就可以对页面上的 DOM 元素进行操作，这让创建动态的和可交互的网页成为可能。我们已经看到了 jQuery 如何帮助动态地创建 DOM 元素。我们可以通过各种方法将这些新元素添加到 DOM 中，也可以移动和复制现有元素。

`append()` 方法可以用来追加新内容到现有内容的末尾。

① <、>和&的实体字符分别是 `<`、`>` 和 `&`。

② 保持元素原先的内容不变，但是可以向其中添加新的元素从而改变其内容。

方法语法：append

append(content)

将传入的 HTML 片段或者元素追加到所有匹配元素的内容中

参数

content	（字符串|元素| jQuery |函数）一个字符串、元素、包装集或者函数，用来指定要添加到包装集元素中的内容。如果参数是一个函数，则为每一个包装元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素的下标和原先的内容。函数的返回值作为要追加的内容

返回值

包装集

这个方法接受的参数包括：包含HTML片段的字符串，对现有或新创建元素的引用，或jQuery元素包装集。

考虑如下的简单情况：

```
$('p').append('<b>some text<b>');
```

这个语句从传入的字符串创建 HTML 片段，将其追加到页面上所有<p>元素的现有内容的末尾。

这个方法更复杂的用法是获取 DOM 中的现有元素作为追加项。考虑如下代码：

```
$("p.appendToMe").append($("a.appendMe"))
```

这个语句将所有拥有 appendMe 类的链接，移动到所有拥有 appendToMe 类的<p>元素的子节点列表的末尾。如果这个操作有多个目标，则会复制必要数量的原始元素副本，并追加到每个目标元素的子节点末尾。所有情况下，原始元素都被从其最初的位置删除了。

如果只找到一个目标元素，那么这种操作就是*移动操作*；原始元素被从其最初的位置删除了，并且出现在目标元素子节点的末尾。如果找到多个目标元素，则这种操作就是复制移动操作，创建足够数量的原始元素副本以便将其追加到每个目标子节点的末尾。

也可以引用指定的 DOM 元素来代替完整的包装集。如下所示：

```
$("p.appendToMe").append(someElement);
```

添加元素到元素内容的末尾是一个常见操作（可能需要添加一个列表项到列表的末尾，添加一行到一个表格的末尾，或者添加一个新元素到文档主体的末尾），然而还可能需要将新建的或现有的元素添加到目标元素内容的*开始*位置。

当这样的需求出现时，prepend()方法就能派上用场了。

方法语法：prepend

prepend(content)

将传入的 HTML 片段或者元素添加到所有匹配元素的内容的开头

参数

content	（字符串|元素|jQuery|函数）一个字符串、元素、包装集或者函数，用来指定要添加到包装集元素中的内容。如果参数是一个函数，则为每一个包装元素调用此函数，设置当前元素为函数上下文，并且传递两个参数：元素的下标和之前的内容。函数的返回值作为要添加的内容

返回值

包装集

有时候，我们可能需要将元素放置到其他地方，而不是元素内容的开头或者末尾。jQuery 允许我们通过识别一个目标元素并将源元素放置到目标元素的前面或者后面，从而把新创建的或者现有的元素放置到 DOM 中的任意位置。

这些方法被命名为 `before()`和 `after()`不足为奇。现在你应该很熟悉它们的语法了。

方法语法：before

before(content)

将传入的 HTML 片段或者元素插入为目标元素的兄弟节点，位于目标元素之前。目标包装元素必须已经是 DOM 的一部分

参数

content	（字符串|元素|jQuery|函数）一个字符串、元素、包装集或者函数，用来指定要添加到 DOM 中目标包装集元素之前的内容。如果参数是一个函数，则为每一个包装元素调用此函数，传递当前元素作为函数上下文。函数的返回值作为要插入的内容

返回值

包装集

这些操作方法对于在页面上高效地使用 DOM 非常关键。我们提供了一个移动与复制实验室页面，你可以利用它多加练习，确保完全理解这些操作方法。这个实验室页面可以从 chapter3/move.and.copy.lab.html 获取，其初始显示如图 3-4 所示。

方法语法：after

after(content)

将传入的 HTML 片段或者元素插入为目标元素的同级节点，位于目标元素之后。目标包装元素必须已经是 DOM 的一部分

参数

content （字符串|元素| jQuery |函数）一个字符串、元素、包装集或者函数，用来指定要添加到 DOM 中目标包装集元素之后的内容。如果参数是一个函数，则为每一个包装元素调用此函数，传递当前元素为函数上下文。函数的返回值作为要插入的内容

返回值

包装集



图 3-4 移动与复制实验室页面可以观察 DOM 操作方法的运行情况

实验室页面左侧的面板包含 3 张图片，可以作为移动/复制测试的源元素。选择图片相应的复选框以选中一张或者多张图片。

移动/复制操作的目标元素在右侧面板中，也是通过复选框选中的。在面板的底部控制按钮允许我们从这四种操作中选择一种来应用：append、prepend、before 或 after。（现在可以先忽略 clone，我们会在后面介绍。）

单击 Execute 按钮会引发这个操作：使用指定的操作将选中的源图片应用到由选中的目标元素集组成的包装集中。上述操作完成后，Execute 按钮将替换成一个 Restore 按钮，我们可以使用这个按钮将一切归位以便运行另一个实验。

接下来运行一个 append 实验。

选中小狗图片，然后再选中 Target 2。选择 append 操作，单击 Execute 按钮。结果如图 3-5 所示。



图 3-5　append 操作使得 Cozmo 被添加到了 Target 2 的尾部

使用移动/复制实验室页面来尝试源、目标，以及这四种操作的各种组合，确保你熟悉它们的运行方式。

有时候，如果能够反转传入到这些操作的元素顺序，那就能使代码更加易读。另一种将元素从一个地方移动或复制到另一个地方的方法，是将源元素包裹起来（而不是目标元素），然后在方法的参数中指定目标元素。噢，jQuery也允许我们这么做，利用与刚探讨过的四种方法类似的操作就可以反转源元素和目标元素的顺序。这些操作是 `appendTo()`、`prependTo()`、`insertBefore()`和 `insertAfter()`，它们的语法如下所示。

方法语法：`appendTo`

`appendTo(targets)`

将包装集中的所有元素追加到指定目标元素内容的末尾

参数

`targets`	（字符串|元素）包含 jQuery 选择器的字符串，或者 DOM 元素。包装集中的所有元素都会被追加到每个目标元素内容的末尾

返回值

包装集

方法语法：`prependTo`

`prependTo(targets)`

将包装集中所有的元素添加到指定目标元素内容的开头

参数

`targets`	（字符串|元素）包含 jQuery 选择器的字符串，或者 DOM 元素。包装集中的所有元素都会被添加到每个目标元素内容的开头

返回值

包装集

方法语法：`insertBefore`

`insertBefore(targets)`

将包装集中的所有元素添加到 DOM 中指定目标元素的前面

参数

`targets`	（字符串|元素）包含 jQuery 选择器的字符串，或者 DOM 元素。包装集中的所有元素都会被添加到每个目标元素的前面

返回值

包装集

方法语法：insertAfter

insertAfter(targets)

将包装集中所有的元素添加到 DOM 中指定目标元素的后面

参数

targets	（字符串|元素）包含 jQuery 选择器的字符串，或者 DOM 元素。包装集中的所有元素都会被添加到每个目标元素的后面

返回值

包装集

在继续学习前我们还需要说明一件事情……

还记得上一章讲过如何使用 jQuery 的`$()`包装器函数来创建新的 HTML 片段吗？当它和 `appendTo()`、`prependTo()`、`insertBefore()`和 `insertAfter()`方法一起使用时，将是非常有用的技巧。考虑如下代码：

```
$('<p>Hi there!</p>').insertAfter('p img');
```

这个语句创建一个友好的段落，并将其副本插入到段落元素内每个图片元素的后面。这是一个习惯用法，我们已经在列表 2-1 中见过，会经常在页面中使用它。

有时候，我们并不想把元素插入到其他元素中，而是需要做相反的操作。下面看看 jQuery 为此提供了哪些功能。

3.3.3　包裹与反包裹元素

另一种经常需要执行的 DOM 操作类型是，在某个标记中包裹一个元素（或一系列元素）。例如，我们可能想要将所有带有某个类的链接包裹到一个`<div>`中。可以使用 jQuery 的 `wrap()`方法来完成这个 DOM 修改操作。它的语法如下所示。

方法语法：wrap

wrap(wrapper)

使用传入的 HTML 标签或元素的副本将匹配集中的元素包裹起来

参数

wrapper	（字符串|元素）一个包含元素开始和结束标签的字符串，用来包裹匹配集中的每个元素；或者一个元素，其副本将用作包装器

返回值

包装集

要在带有 hello 类的`<div>`中将所有拥有 surprise 类的链接分别包裹起来，可以这么写：

```
$("a.surprise").wrap("<div class='hello'></div>")
```

如果想使用页面上的第一个<div>元素的副本将这些链接包裹起来，可以这么写：

```
$("a.surprise").wrap($("div:first")[0]);
```

如果包装集中有多个元素，则 wrap()方法会分别应用到每个元素上。如果想要将包装集中的全部元素作为一个整体包裹起来，则可以改用 wrapAll()方法：

方法语法：wrapAll

wrapAll(wrapper)

使用传入的 HTML 标签或者元素的副本，将匹配集中的元素作为一个整体包裹起来

参数

wrapper （字符串|元素）一个包含元素开始和结束标签的字符串，用来包裹匹配集中的每个元素；或者一个元素，其副本将用作包装器

返回值

包装集

有时候，我们想包裹的不是匹配集中的元素，而是这些元素的内容。在这种情况下，就可以使用 wrapInner()方法。

方法语法：wrapInner

wrapInner(wrapper)

使用传入的 HTML 标签或者元素的副本，将匹配集中的元素内容（包含文本节点）包裹起来

参数

wrapper （字符串|元素）一个包含元素开始和结束标签的字符串，用来包裹匹配集中的每个元素；或者一个元素，其副本将用作包装器

返回值

包装集

其反向操作（也就是删除子元素的父节点）可以通过 unwrap()方法来完成。

方法语法：unwrap

unwrap()

删除包装元素的父元素。子元素和其所有的同级节点一起替换了 DOM 中的父元素

参数

无

返回值

包装集

现在，我们已经知道了如何创建、包裹、反包裹、复制和移动元素，那如何删除页面上的元素呢？

3.3.4　删除元素

和添加、移动、复制 DOM 中的元素同样重要的功能是删除不再需要的元素。

如果想清空或者删除一组元素，可以使用 `remove()`方法，它的语法如下所示。

方法语法：`remove`

`remove(selector)`

从页面 DOM 中删除包装集中的所有元素

参数

`selector`　（字符串）一个可选的选择器，可以用来进一步过滤包装集中要删除的元素

返回值

包装集

注意，和很多其他的 jQuery 方法一样，这个方法的返回结果也是包装集。从 DOM 中删除的元素依然可以通过这个集合来引用（因此不符合垃圾回收条件），可以使用其他 jQuery 方法进一步操作，包括 `appendTo()`、`prependTo()`、`insertBefore()`、`insertAfter()`以及其他类似的方法。

尽管如此，在使用 `remove()`从 DOM 中删除元素时要注意，绑定到元素上的任何 jQuery 数据或事件也会被同时删除。`detch()`与此类似，也可以用来从 DOM 中删除元素，但是会保留绑定的事件和数据。

方法语法：`detach`

`detach(selector)`

从页面 DOM 中删除包装集中的所有元素，保留绑定的事件和 jQuery 数据

参数

`selector`　（字符串）一个可选的选择器字符串，用来进一步过滤包装集中要删除的元素

返回值

包装集

如果想在删除元素后，再把它放回到 DOM 中并且保持其事件和数据的完整，那么首选 `detach()`方法。

使用 `empty()`方法可以完全清空 DOM 元素的内容，其语法如下所示。

方法语法：empty

empty()
删除匹配集中所有 DOM 元素的内容

参数

无

返回值

包装集

有时候，我们不想移动元素，而是想要复制它们……

3.3.5 复制元素

操作 DOM 的另外一种方式是创建元素的副本，并把它们添加到 DOM 树的其他地方。jQuery 提供了一个方便的包装器方法 clone()来完成上述操作。

方法语法：clone

clone(copyHandlers)
创建包装集中元素的副本，并返回包含这些副本的新包装集。这些元素和任何子节点都会被复制。是否复制事件处理器取决于 copyHandlers 参数的设置

参数

copyHandlers	（布尔）如果参数是 true，则复制事件处理器。如果参数是 false 或者被省略，则不复制事件处理器

返回值

新建的包装集

使用 clone()创建现有元素的副本用处并不是很大，除非我们对那些副本进行一些操作。一般来说，一旦生成包含副本的包装集，我们就可以应用另一个 jQuery 方法把它们放置到 DOM 中的某个地方。例如：

```
$('img').clone().appendTo('fieldset.photo');
```

这个语句创建所有图片元素的副本，并把它们追加到所有带有类名 photo 的<fieldset>元素中。

一个稍微复杂的例子如下所示：

```
$('ul').clone().insertBefore('#here');
```

这个方法链执行类似的操作，但是复制操作的目标（所有的<ul>元素）都会被复制，包括其子节点（任何<ul>元素都有可能包含一些<li>子节点）。

最后一个例子：

```
$('ul').clone().insertBefore('#here').end().hide();
```

这个语句执行的操作和前一个例子相同，但是在插入副本之后，会利用 end()方法来选择原始包装集（原始的目标元素）并隐藏它们。这个示例阐明了复制操作如何创建包含一组新的元素的包装集。

为了动态地查看复制操作，请返回到移动与复制实验室页面。在 Execute 按钮的上方是一对单选按钮，它允许我们将复制操作作为主要的 DOM 处理操作的一部分。当选中 yes 单选按钮时，源元素会在 append、prepend、before 或者 after 方法执行之前被复制。

在启用复制的情况下，重复一些之前进行的实验，注意观察这些操作是如何影响原始的源元素的。

我们可以插入、删除和复制元素。组合使用这些操作可以很容易地完成更高级的操作，比如替换。不过这其实没有你想的那么麻烦。

3.3.6　替换元素

有时候，我们需要使用新元素来替换现有元素，或者移动现有元素以取代另一个元素，jQuery 为此提供了 replaceWith()方法。

方法语法：replaceWith

replaceWith(content)

使用指定的内容替换每个匹配元素

参数

content	（字符串|元素|函数）一个包含 HTML 片段的字符串，作为将要替换的内容；或者一个元素的引用，它会被移动并替换现有的元素。如果参数是一个函数，则为每个包装元素分别调用此函数，设置当前元素为函数上下文，并且不传递任何参数。函数的返回值作为新的内容

返回值

包含被替换元素的 jQuery 包装集

假设在特定情况下，我们希望使用包含图片 alt 值的<span>元素来替换页面上所有拥有 alt 值的图片。使用 each()和 replaceWith()可以这样来实现：

```
$('img[alt]').each(function(){
  $(this).replaceWith('<span>'+ $(this).attr('alt') +'</span>')
});
```

each()方法允许我们遍历每一个匹配元素，replaceWith()将图片替换为生成的<span>元素。

如果我们想继续使用这些元素而不是直接丢弃它们，则可以使用 replaceWith()方法返回一个包含从 DOM 中删除的元素的 jQuery 包装集。作为练习，请思考如何扩展这个示例代码以便将那些已删除的元素重新添加到 DOM 中的其他地方。

当向 replaceWith()传入一个现有元素时，该元素首先会从 DOM 中的原始位置删除，然后再重新添加以替换目标元素。如果有多个这样的目标元素，则会复制足够数量的原始元素副本。

有时，颠倒 replaceWith()指定的元素顺序会很方便，以便使用匹配选择器指定用来替换的元素。我们已经看到过类似的互补的方法，比如 append()和 appendTo()，它们允许我们指定最适合代码的元素顺序。

类似地，replaceWith()方法的镜像方法是 replaceAll()，它允许我们执行类似的操作，但是以相反的顺序来指定元素。

方法语法：replaceAll

replaceAll(selector)

使用调用此方法的匹配集的内容，替换所有与传入的选择器相匹配的元素

参数

selector （选择器）一个选择器字符串表达式，用来识别要被替换的元素

返回值

包含插入元素的 jQuery 包装集

和 replaceWith()一样，replaceAll()也返回一个 jQuery 包装集。但是这个集合包含的不是被替换掉的元素，而是用来替换的元素。被替换的元素已经丢失，因此不能对其进行进一步操作。当你决定使用哪种替换方法时，一定要考虑到这一点。

现在，我们已经讨论了如何处理一般的 DOM 元素，接下来简单看下如何处理一种特殊类型的元素：表单元素。

3.4 处理表单元素值

因为表单元素拥有特殊的属性，所以 jQuery 核心包含很多便捷的函数来完成如下操作：

- 获取和设置表单元素的值；
- 序列化表单元素；
- 基于表单特定的属性选择元素。

这些函数在大部分情况下可以很好地为我们服务，不过表单插件（一个由 jQuery 核心团队成员开发的官方支持的插件）提供了更多与表单相关的功能。有关这个插件的更多内容可以从这里获取：http://jquery.malsup.com/form/。

什么是表单元素

当我们使用术语“表单元素”时，指的是那些出现在表单内部的元素，拥有 name 和 value 特性，并且在提交表单时，它们的值会作为 HTTP 请求参数发送到服务器。在脚本中手工处理

> 这些元素是需要技巧的，因为不仅可以禁用元素，而且 W3C 还为控件定义了一个失败的状态[1]。这个状态决定了在提交过程中应该忽略哪些元素，而这仅仅是复杂面的一小部分。

看看在表单元素上进行的最常见的一种操作：访问它的值。jQuery 的 `val()`方法考虑了最常见的情况，返回包装集中的第一个表单元素的 `value` 特性。其语法如下所示。

方法语法：val

val()

返回匹配集中第一个元素的 value 特性。如果此元素是一个可以多选的元素时，返回值是所有选择项所组成的数组[2]

参数

无

返回值

获取的值

虽然这个方法非常有用，但是它也有一些限制需要我们注意。如果包装集中的第一个元素不是一个表单元素，则会返回一个空字符串，这并不是我们真正需要的值（`undefined` 或许会更清楚些）。这个方法也不区分复选框和单选按钮的选中或非选中状态，只是简单地返回由它们的 `value` 特性所定义的值，而无论它们是否被选中。

对于单选按钮，组合使用 jQuery 选择器和 `val()`方法的能力足够扭转局面，我们已经在本书的第一个例子中看到过。考虑包含名为 `radioGroup` 的单选按钮分组（一组名称相同的单选按钮）的表单，以及下面的表达式：

```
$('[name="radioGroup"]:checked').val()
```

这个表达式返回唯一选中的单选按钮的值（如果没有选中任何单选按钮，则返回 `undefined`）。这比遍历按钮来查找选中的元素简单多了，不是吗？

由于 `val()`只考虑包装集中的第一个元素，因此并不适用于可能选中多个控件的复选框分组。然而 jQuery 是不会让我们孤立无援的。考虑如下代码：

```
var checkboxValues = $('[name="checkboxGroup"]:checked').map(
  function(){ return $(this).val(); }
).toArray();
```

尽管我们还没有正式学习如何扩展 jQuery（详见第 7 章），不过你已经看了很多示例代码，不妨小试牛刀。看看你能否将前面的代码重构成一个 jQuery 包装器方法，用来返回由包装集中

① 关于表单中哪些控件是可以提交的，请参考 W3C 网站 http://www.w3.org/TR/html4/interact/forms.html 的 13.2 节的内容。

② 比如`<select id="list" multiple="multiple">`，则表达式`$("#list").val()`返回包含一个或多个元素的数组。

任何选中的复选框组成的数组。

虽然 val()方法能够非常出色地获取任何单个的表单控件元素的值，但是如果希望通过提交表单来获取一套完整的值，最好使用 serialize()和 serializeArray()方法（详见第 8 章）或者官方表单插件。

另一个常见操作是设置一个表单元素的值。只需提供一个值，val()方法就可以实现这个目的。其语法如下所示。

3

方法语法：val

val(value)

设置传入的值为所有匹配的表单元素的 value

参数

value	（字符串|函数）指定一个值，将其设置为包装集中每个表单元素的 value 属性。如果参数是个函数，则为包装集中每个元素分别调用此函数，设置当前元素为函数上下文，传递两个参数：元素下标和元素的当前值。函数的返回值作为要设置的值

返回值

包装集

val()方法的另一个用途是选中复选框或者单选按钮元素，或者选择<select>元素中的可选项。val()的这个变体的语法如下所示。

方法语法：val

val(values)

导致包装集中任何复选框、单选按钮或者<select>元素的可选项变成选中（checked）或者选择（selected）状态，只要它们值匹配传入的 values 数组中的任何一个值

参数

values	（数组）一个数组，用来确定应该选中或者选择哪些元素

返回值

包装集

考虑下面这个语句：

```
$('input,select').val(['one','two','three']);
```

这个语句会搜索页面上所有的<input>和<select>元素，只要这些元素的值与输入字符串'one'、'two'或者'three'任何一个相匹配。任何匹配的复选框或者单选按钮都会变成选中状态，而任何匹配的可选项将会变为选择状态。

这使得 `val()`的用途不仅仅局限于处理基于文本的表单元素。

3.5　小结

在本章中，我们超越了选择元素的技术并且开始操作元素。利用目前所学的技术，我们可以使用强大的条件来选择元素，然后将它们移动到页面的任意部位，就像做外科手术那样。

我们可以选择复制、移动、替换元素，甚至从头创建全新的元素。可以在页面上追加、前置、包裹任意元素或者元素集。我们还学习了如何管理表单元素的值，所有这一切都通向强大而简洁的逻辑。

有了这些基础知识，我们准备深入学习更多的高级概念，下面就从典型的、棘手的页面事件处理开始。

第 4 章

事件处理

4

本章内容

- ❑ 浏览器实现的事件模型
- ❑ jQuery 事件模型
- ❑ 为 DOM 元素绑定事件处理器
- ❑ `Event` 对象实例
- ❑ 在脚本控制下触发事件处理器
- ❑ 预先注册事件处理器

熟悉百老汇的歌舞剧 Cabaret《歌厅》以及其同名好莱坞电影的任何人，可能会记得“Money Makes the World Go Around”（有钱能使鬼推磨）这首歌。这一金钱万能的观点适用于物质世界，在虚拟的万维网王国里，正是事件使这一切运行起来的！

类似于其他 GUI 管理系统，HTML 网页呈现的界面是异步的和事件驱动的（即使用来将页面传送给浏览器的 HTTP 协议本质上是完全同步的）。无论 GUI 是使用 Java Swing、X11 或者.NET Framework 实现的桌面程序，还是 Web 应用中使用 HTML 和 JavaScript 实现的网页，程序的步骤基本上都是一样的：

(1) 建立用户界面；

(2) 等待有趣的事情发生；

(3) 做出相应的反应；

(4) 转到第(2)步。

步骤(1)创建用户界面的*显示*，其他步骤定义用户界面的*行为*。在 Web 页面中，浏览器负责创建显示界面来响应发送给它的标记（HTML 和 CSS）。网页中包含的脚本定义了显示界面的行为。

这些脚本以*事件处理器*的形式存在（也被称为*监听器*），它们负责对页面显示时发生的各种事件做出响应。这些事件可能是系统生成的（比如定时器或者完成异步请求），但是大部分是用户操作的结果（比如移动或单击鼠标，通过键盘输入文本，甚至是 iPhone 的手势）。如果没有对这些事件的响应能力，万维网的最大用途可能会仅限于显示一些小猫的照片。

虽然 HTML 本身*确实*定义了少数几个无需我们编写脚本来实现的内置的语义动作（比如单击锚点标记（`<a>`）后重新加载页面或者通过单击提交按钮来提交表单），但是要想让页面展示任

何其他行为，就需要处理用户和页面交互时产生的各种事件。

在本章中，我们将探讨浏览器公开这些事件的各种方式，如何创建处理器以便对发生的事件做出响应，以及如何应对由于浏览器事件模型的差异而带来的诸多挑战。然后，我们将看到 jQuery 是如何穿透这层由浏览器而导致的迷雾，从而帮助我们分担重负的。

下面开始研究浏览器如何公开其事件模型。

你需要知道的 JavaScript

jQuery 为 Web 应用开发带来的一个最大的好处就是，无需动手编写一大堆脚本就能实现大量的脚本行为。jQuery 处理了具体的细节，因此只需关注 Web 应用需要实现的功能即可！

目前为止，我们的学习之旅是相当顺利的。只需基本的 JavaScript 技能就能编写和理解前面几章介绍的 jQuery 示例。但在本章和随后的几章中，你必须理解几个重要的 JavaScript 基础概念才能有效地使用 jQuery 库。

可能由于你的技术背景，你已经熟悉了这些概念，但即便是有些页面开发者没能牢固地掌握这些概念，他们也能够编写出大量的代码——一切有赖于 JavaScript 的高度灵活性。在继续学习之前，应该确保你已经牢牢掌握了这些核心概念。

如果已经熟悉了 JavaScript 对象和函数类的工作方式，并且也充分了解函数上下文和闭包的概念，那么你可以继续阅读本章及接下来的几章。如果对这些概念感到陌生或者模糊的话，强烈建议你转到附录，它可以帮助你加速理解这些必备的概念。

4.1　浏览器的事件模型

早在有人考虑如何标准化浏览器事件处理之前，网景通信公司（Netscape Communications Corporation）就在其网景领航员（Netscape Navigator）浏览器中引入了一个事件处理模型。所有的现代浏览器依然支持这种模型，它可能是大多数网页开发者理解最深入、应用最广泛的事件处理模型。

这个模型有多种名称。你也许听过它被称为网景事件模型（Netscape Event Model）、基本事件模型（Basic Event Model），甚至它被含糊地称为浏览器事件模型（Browser Event Model），但是大部分人称其为 DOM 第 0 级事件模型（DOM Level 0 Event Model）。

注意　术语 DOM 级别（DOM Level）用来表示 W3C DOM 规范的实现达到了什么级别的需求。不存在 DOM 第 0 级，这个术语是用来非正式地描述 DOM 第 1 级之前实现的那些规范。

直到 2000 年 11 月 DOM 第 2 级被引入时，W3C 才真正为事件处理建立了标准模型。这个模型得到了所有标准兼容的现代浏览器的支持，比如 Firefox、Camino（以及其他的 Mozilla 浏览器），Safari 和 Opera。IE 浏览器则继续特立独行，它支持 DOM 第 2 级事件模型的一个功能子集——尽管使用的是专有接口。

在了解 jQuery 如何化解这个恼人的事实之前，先花一些时间学习各种事件模型是如何运作的。

4.1.1 DOM第 0 级事件模型

DOM 第 0 级事件模型是大多数 Web 开发者在他们的页面上采用的事件模型。除了有点依赖于浏览器之外，它使用起来还是相当容易的。

在这种事件模型下，事件处理器是通过将一个函数实例的引用赋值给 DOM 元素的属性来声明的。定义这些属性来处理特定的事件类型。例如，将函数赋值给 onclick 属性来处理单击事件，将函数赋值给 onmouseover 属性来处理 mouseover 事件，而这些元素需要支持这些事件类型。

浏览器允许将事件处理函数的主体作为特性值嵌入到 DOM 元素的 HTML 标记中，这样提供了创建事件处理器的便捷方式。定义这种处理器的示例如代码清单 4-1 所示。这个页面可以在本书的示例代码文件 chapter4/dom.0.events.html 中找到。

代码清单 4-1 声明 DOM 第 0 级事件处理器

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Level 0 Events Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.min.js"></
     script>
    <script type="text/javascript" src="../scripts/jqia2.support.js"></
     script>
    <script type="text/javascript">
      $(function(){
        $('#example')[0].onmouseover = function(event) {      ❶ 定义 mouseover 处理器
          say('Crackle!');                                      ❷ 输出文本到“控制台”
        };
      });
    </script>
  </head>

  <body>
    <img id="example" src="example.jpg"                         ❸ 设置<img>元素
         onclick="say('BOOM!');"
         alt="ooooh! ahhhh!"/>
  </body>
</html>
```

本例使用了两种方式来声明事件处理器：在脚本控制下声明和在标记特性中声明。

页面首先定义了一个就绪处理器，在其中获取 id 为 example 的图片元素的引用（使用 jQuery），并设置它的 onmouseover 属性为内联函数❶。当 mouseover 事件在元素上触发时，这个函数就成为了它的事件处理器。注意，这个函数要求传入一个参数。我们将很快学到有关这个参数的更多知识。

在这个函数内，我们使用了一个小的实用函数服务（say()❷），用来将文本信息输出到页面上一个动态生成的<div>元素中（称为“控制台”）。这个函数在导入的支持脚本文件（jqia2.support.js）中声明，当页面上有事件发生时，这个函数可以减少一些麻烦，我们不必使用令人讨厌的和具有破坏性的警告框来指示有事件发生。我们将在本书后面部分的很多示例中使用这个方便的函数。

在页面的主体（<body>）中，我们放置了一个在其中定义事件处理器的<img>元素。我们已经看到在就绪处理器中如何在脚本控制下定义处理器❶，不过这里使用<img>元素的 onclick 特性❸声明了一个单击事件处理器。

注意　显然，在本例中我们抛弃了不唐突的 JavaScript 概念。在完成的本章学习后，我们将看到为什么再也不需要将事件行为嵌入到 DOM 标记中去了！

在浏览器中打开这个页面（找到文件 chapter4/dom.0.events.html），在图片上移动几次鼠标指针，然后单击这个图片，显示的结果类似于图 4-1 所示。

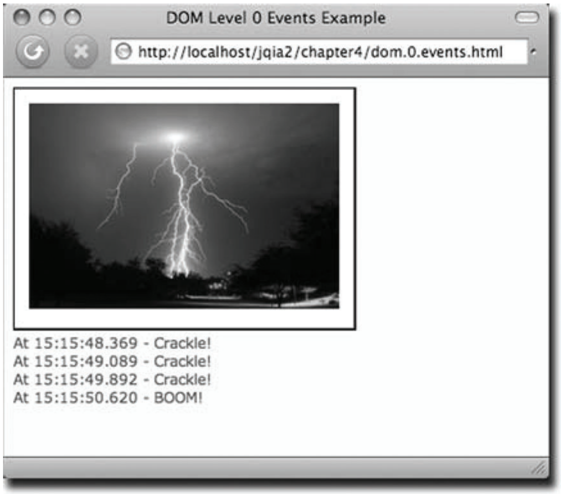


图 4-1　在图片上移动鼠标并单击图片，触发事件处理器——输出消息到控制台

我们在<img>元素标记中使用如下特性声明单击事件处理器：

```
onclick="say('BOOM!');"
```

这可能让人误以为 say()函数成为了该元素的单击事件处理器，但事实并非如此。当通过 HTML 标记特性声明处理器时，会自动创建一个将特性值作为函数体的匿名函数。假设 imageElement 是对此图片元素的引用，下面的代码等价于通过特性声明创建的结构：

```
imageElement.onclick = function(event) {
  say('BOOM!');
};
```

注意，特性值是如何作为生成的函数的函数体的，并且注意由于创建了函数，从而可在其中使用 event 参数。

在继续探索什么是 event 参数之前，应该注意到，使用特性机制来声明 DOM 第 0 级事件处理器违反了第 1 章中探讨的不唐突的 JavaScript 原则。当在页面中使用 jQuery 时，应该遵守不唐突的 JavaScript 原则，避免把显示标记与 JavaScript 定义的行为混合使用。我们很快就会看到，jQuery 所提供的声明事件处理器的方式要比这两种方式更好。

不过，首先探讨下什么是 event 参数。

1. Event 实例

在大部分浏览器中，当一个事件处理器被触发时，名为 Event 的类实例会作为第一个参数传入处理器中。而一直占据主流地位的 IE 却以自己专有的方式行事，将 Event 实例保存到一个名为 event 的全局属性中（也就是 window 的一个属性）。

为了处理这种差异，我们经常会在非 jQuery 的事件处理器的第一行看到如下代码：

```
if (!event) event = window.event;
```

这个语句通过特征检测（详见第 6 章）来检查 event 参数是否为 undefined（或者 null），如果是的话就把 window 的 event 属性赋值给它，从而消除了浏览器差异。在这个语句之后，可以在处理器中随意引用 event 参数，无需考虑它是如何在处理器中变得可用的。

Event 实例属性提供了关于当前正在被处理的已触发事件的大量信息。这包括一些细节，比如在哪个元素上触发的事件、鼠标事件的坐标以及键盘事件中单击了哪个键。

先别急，IE 浏览器不仅使用专有的方法来获取处理器使用的 Event 实例，而且还使用专有的 Event 类定义来取代 W3C 定义的标准——我们还没有逃出对象检测的丛林。

例如，为了获取目标元素（正是在该元素上，事件被触发）的引用，在标准兼容的浏览器中使用 target 属性，在 IE 中则使用 srcElement 属性。通过对象检测来处理这种不一致性，代码如下所示：

```
var target = (event.target) ? event.target : event.srcElement;
```

这个语句检查 event.target 的定义是否存在，如果存在，就把它的值赋值给局部变量 target；否则，将 event.srcElement 赋值给 target。对于其他 Event 属性，也需要采取类似的步骤。

到目前为止，似乎事件处理器只与触发事件的元素相关（例如代码清单 4-1 中的图片元素），但是事件会传播到整个 DOM 树。下面来探讨这个主题。

2. 事件冒泡

当触发了 DOM 树中一个元素上的事件时，浏览器的事件处理机制会检查这个元素上是否已经创建了特定的事件处理器。如果是，就会调用已创建的事件处理器。但是事情还没有结束。

在目标元素有机会处理事件后，事件模型会检查目标元素的父元素，看其是否已经为此事件类型创建了处理器。如果是，就调用已创建的处理器，之后检查它的父元素，以及父元素的父元素，以此类推，直到 DOM 树的顶部。因为事件处理的向上传播就像香槟酒杯里冒起的气泡一样（假设 DOM 树的根在顶部），所以整个过程被称为事件冒泡。

下面修改代码清单 4-1 中的示例，以便动态地观察这个过程的运行。考虑代码清单 4-2。

代码清单 4-2　事件从源向上传播到 DOM 的顶部

```
<!DOCTYPE html>
<html id="greatgreatgrandpa">
  <head>
    <title>DOM Level 0 Bubbling Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript" src="../scripts/jqia2.support.js"></
     script>
    <script type="text/javascript">
      $(function(){
        $('*').each(function(){                    ← ❶ 选择页面上的所有元素
          var current = this;
          this.onclick = function(event) {          ← ❷ 为选中的每一个元素应用 onclick 处理器
            if (!event) event = window.event;
            var target = (event.target) ?
                          event.target : event.srcElement;
            say('For ' + current.tagName + '#'+ current.id +
                ' target is ' +
                target.tagName + '#' + target.id);
          };
        });
      });
    </script>
  </head>

  <body id="greatgrandpa">
    <div id="grandpa">
      <div id="pops">
        <img id="example" src="example.jpg" alt="ooooh! ahhhh!"/>
      </div>
    </div>
  </body>
</html>
```

本例中，我们做了很多有趣的改动。首先，删除了之前的 `mouseover` 事件处理器，以便将注意力集中于单击事件。其次把事件实验目标的图片元素插入到两个嵌套的`<div>`元素中，目的是为了将图片元素放在 DOM 层次结构中的深处。我们也给页面中几乎每个元素赋予了一个特定的和唯一的 `id`——甚至包括`<body>`和`<html>`标签！

现在来看看更加有趣的改动。

在页面的就绪处理器中，我们使用 jQuery 选择页面上的所有元素，并使用 `each()`方法❶遍历每一个元素。对于每个匹配的元素，我们将它的实例保存在局部变量 `current` 中，并创建了 `onclick` 处理器❷。这个处理器首先使用上节讨论过的依赖于浏览器的技巧，找到 `Event` 实例并识别事件目标，然后输出一条控制台消息。这条消息是本例中最有趣的部分。

本例使用闭包（如果闭包这一主题让你感到很迷惑，请阅读 A.2.4 节）来显示当前元素的标签名称和 `id`，后面紧跟着目标元素的 `id`。这样做使得输出到控制台的每一个消息都能展示冒泡过程中当前元素的信息，以及引发整个过程的目标元素的信息。

加载页面（文件 chapter4/dom.0.propagation.html）并单击图片，结果如图 4-2 所示。

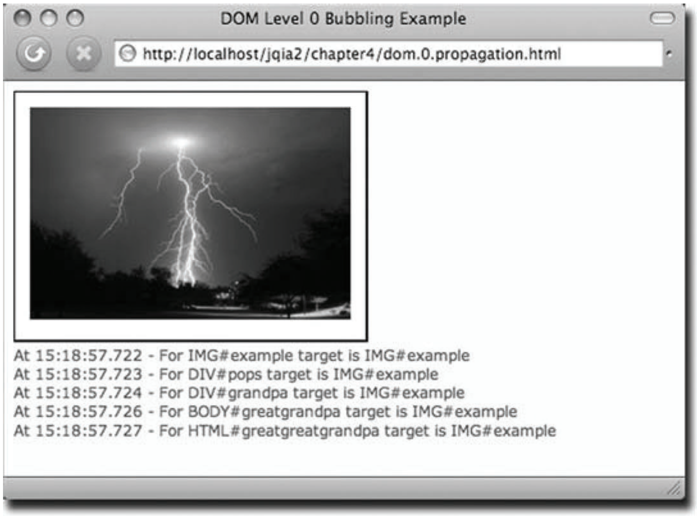


图 4-2 控制台消息清楚地展示了事件的传播过程，它从目标元素开始一直冒泡到 DOM 树的根元素

这清楚地说明了事件触发的时候，事件首先传递到目标元素上，然后按顺序传递到各祖先元素上，直到根元素`<html>`为止。

这是一个强大的功能，因为它允许我们在任何层级的元素上创建处理器，以便处理发生在后代元素上的事件。比如说有一个定义在`<form>`元素上的处理器，它可以用来响应在该元素的任何后代元素上发生改变的事件，从而根据后代元素的新值来动态地调整显示。

但是如果不想让事件传播该怎么办？能不能阻止事件传播？

3. 影响事件传播和语义动作

在有些场合，我们可能希望阻止事件继续向 DOM 树的上层冒泡。这可能是因为我们态度严谨并且知道已经完成了对事件的必要处理，或者想防止可能在作用链的高层中发生的不必要的处理。

无论何种原因，都可以通过 `Event` 实例提供的机制来阻止事件向更高层传播。对于标准兼容的浏览器，可以调用 `Event` 实例的 `stopPropagation()`方法来阻止事件向更高层的祖先元素传播。对于 IE 浏览器，可以设置 `Event` 实例的属性 `cancelBubble` 的值为 `true`。有趣的是，很多现代标准兼容的浏览器也支持 `cancelBubble` 机制，尽管它不是任何 W3C 标准的一部分。

有些事件有关联的默认语义。例如，锚点元素（`<a>`）上的单击事件将导致浏览器重定向到元素的 `href` 特性值，`<form>`元素的提交事件将导致表单提交。如果希望取消这些语义动作（有时称为默认动作），那么可以将事件处理器的返回值设为 `false`。

这个操作在表单验证领域很常用。在表单的提交事件处理器中，可以对表单控件进行验证，如果检测到任何数据输入项存在问题就返回 `false`。

你可能已经在`<form>`元素上见过如下代码：

```
<form name="myForm" onsubmit="return false;" ...
```

在任何情况下这都可以有效地防止表单被提交，除非是在脚本控制下的提交（通过 form.submit()来提交不会触发提交事件）。在许多使用异步请求来代替表单提交的 Ajax 应用中，这是一个常用技巧。

在 DOM 第 0 级事件模型中，事件处理器中的几乎每个步骤都需要使用浏览器特定检测，目的是为了确定下一步的动作。这是多么令人头疼的事！但是别丢掉热情和耐心——在考虑更加高级的事件模型时，事情不会更简单。

4.1.2　DOM第 2 级事件模型

DOM 第 0 级事件模型有一个严重的缺点，因为存储在属性上的函数引用充当了事件处理器，所以对于任意指定的事件类型，一个元素每次只能注册一个事件处理器。如果希望在单击元素时执行两件事情，下面的代码实现不了这一目的：

```
someElement.onclick = doFirstThing;
someElement.onclick = doSecondThing;
```

因为第二个赋值语句替换掉了前面设置的 onclick 属性值，所以当触发事件时只调用了 doSecondThing。当然，可以使用另一个函数来包装这两个函数，这样就可以通过这个函数调用这两个函数，但是随着页面变得更复杂，尤其在高交互性的应用中，保持对此类事情的跟踪将越来越困难。此外，如果在页面中使用多个可重用的组件或库，它们可能根本不知道其他组件的事件处理需求。

我们可以引入其他解决方案：实现观察者模式，为处理器创建一个发布/订阅方案，或者使用闭包的技巧。但是所有的这些都会增加页面的复杂度，而这些页面本来可能就已经很复杂了。

除了建立标准的事件模型外，DOM 第 2 级事件模型被设计来处理这类问题。下面来看看在这个更高级的事件模型下如何为 DOM 元素创建事件处理器，甚至是多个处理器。

1. 建立事件处理器

DOM 第 2 级事件处理器（也称为监听器）是通过元素方法创建的，而不是把函数引用赋值给元素属性。每个元素都定义有一个名为 addEventListener()的方法，用来附加事件处理器（监听器）到元素上。这个方法的格式如下：

```
addEventListener(eventType,listener,useCapture)
```

eventType 参数是个字符串，用来获取需要处理的事件类型。这些字符串值通常和 DOM 第 0 级事件模型中不带 on 前缀的事件名称是一样的。例如，click、mouseover、keydown 等。

listener 参数是一个函数引用（或者内联函数），用来在元素上创建指定事件类型的处理器。正如在基本事件模型里一样，Event 实例是传入这个函数的第一个参数。

最后一个参数 useCapture 是布尔类型，在随后讨论第 2 级模型的事件传播时会对其进行探讨。目前将其设置为 false。

再次修改代码清单 4-1 中的示例以便使用更高级的事件模型。我们只关注单击（click）事件类型，这次为图片元素创建 3 个单击事件处理器。新的示例代码可以在文件 chapter4/dom.2.events.html 中找到，内容如代码清单 4-3 所示。

代码清单 4-3 使用 DOM 第 2 级事件模型创建事件处理器

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Level 2 Events Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript" src="../scripts/jqia2.support.js">
    </script>
    <script type="text/javascript">
      $(function(){                                    ❶ 创建 3 个事件处理器
        var element = $('#example')[0];
        element.addEventListener('click',function(event) {
          say('BOOM once!');
        },false);
        element.addEventListener('click',function(event) {
          say('BOOM twice!');
        },false);
        element.addEventListener('click',function(event) {
          say('BOOM three times!');
        },false);
      });
    </script>
  </head>

  <body>
    <img id="example" src="example.jpg"/>
  </body>
</html>
```

这段代码非常简单，但是它清楚地展示了如何在同一个元素上为同一个事件类型创建多个事件处理器——使用基本事件模型是不可能轻松做到这一点的。在页面的就绪处理器中❶获取图片元素的引用，然后为单击事件创建 3 个事件处理器。

在标准兼容的浏览器（非 IE 浏览器）中加载这个页面，然后单击图片，结果如图 4-3 所示。

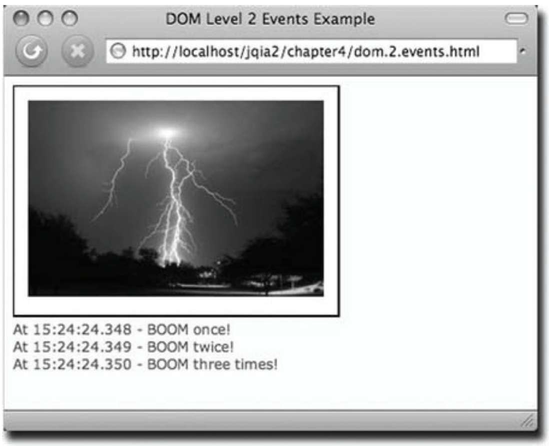


图 4-3 单击一次图片，触发了为单击事件创建的 3 个处理器

注意，虽然这些处理器按照创建的顺序触发，但是标准并不保证这种顺序！代码的测试者从未观察到不同于创建时顺序的情况出现，但是编写依赖于这种顺序的代码是愚蠢的。请时刻注意，在一个元素上创建的多个处理器可能会以随机的顺序触发。

现在看看 `useCapture` 参数的作用。

2. 事件传播

前面我们看到，使用基本事件模型，一旦在元素上触发事件，这个事件会从目标元素沿着 DOM 树向上传播到所有的祖先元素。高级的第 2 级事件模型也提供了这种冒泡阶段，并且还增加了额外的捕获阶段。

在 DOM 第 2 级事件模型中，当触发一个事件时，事件首先从 DOM 树的根部向下传播到目标元素，然后再次从目标元素向上传播到 DOM 树根部。前一个阶段（根到目标）称为捕获阶段，后一个阶段（目标到根）称为冒泡阶段。

当把函数创建为事件处理器时，可以将其标记为捕获处理器，这种情况下会在捕获阶段触发事件，也可以将其标记为在冒泡阶段触发事件的冒泡处理器。此时你可能已经猜到，`addEventListener()`的 `useCapture` 参数是用来识别创建的是什么类型的处理器。将此参数设置为 `false` 时，创建冒泡型处理器，反之参数为 `true` 则建立捕获处理器。

回想一下代码清单 4-2 中的示例，在那里我们探索了基本事件模型下事件在 DOM 层次结构中的传播。在那个示例中，我们将一个图片元素嵌入到两层`<div>`元素中。在这种层次结构中，以`<img>`元素为目标的单击事件在 DOM 树中的传播路径如图 4-4 所示。

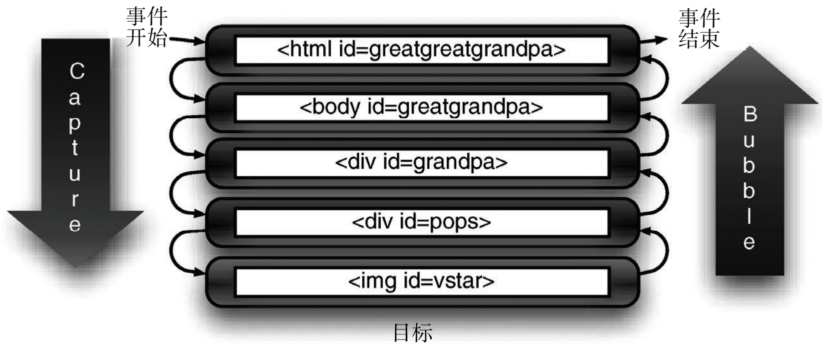


图 4-4　在 DOM 第 2 级事件模型中，事件在 DOM 层次结构中有两次传播过程：一次是从顶部到目标的捕获阶段，另一次是从目标到顶部的冒泡阶段

现在就来测试一下吧。代码清单 4-4 显示的页面代码包含图 4-4 所示的元素层次结构（chapter4/dom.2.propagation.html）。

代码清单 4-4　使用冒泡和捕获处理器跟踪事件的传播

```
<!DOCTYPE html>
<html id="greatgreatgrandpa">
  <head>
    <title>DOM Level 2 Propagation Example</title>
```

```
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript" src="../scripts/jqia2.support.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('*').each(function(){                    ❶ 为所有元素创建监听器
          var current = this;
          this.addEventListener('click',function(event) {
            say('Capture for ' + current.tagName + '#'+ current.id +
                ' target is ' + event.target.id);
            },true);
          this.addEventListener('click',function(event) {
            say('Bubble for ' + current.tagName + '#'+ current.id +
                ' target is ' + event.target.id);
            },false);
        });
      });
    </script>
  </head>

  <body id="greatgrandpa">
    <div id="grandpa">
      <div id="pops">
        <img id="example" src="example.jpg"/>
      </div>
    </div>
    <div id="console"></div>
  </body>
</html>
```

这段代码修改代码清单 4-2，使用 DOM 第 2 级事件模型 API 来创建事件处理器。在就绪处理器❶中，我们利用 jQuery 强大的能力遍历 DOM 树中的所有元素。在每个元素上创建了两个处理器：一个捕获处理器和一个冒泡处理器。每个处理器都输出一条消息到控制台，来识别处理器的类型、当前元素以及目标元素的 `id`。

在标准兼容的浏览器中加载此页面，单击图片的结果如图 4-5 所示，展示了事件在不同处理阶段通过 DOM 树的传播过程。注意，因为我们为目标元素定义了捕获处理器和冒泡处理器，所以这两种处理器会在目标元素和其所有祖先节点上执行。

既然我们已经克服了所有困难理解了这两种类型的处理器，就应该知道捕获处理器几乎不在 Web 页面中使用。原因很简单，就是 IE 浏览器不支持 DOM 第 2 级事件模型。尽管 IE 浏览器确实拥有一个与第 2 级标准的冒泡阶段相对应的专有模型，但是它不支持任何类似的捕获阶段。

在看 jQuery 如何帮助我们收拾这个烂摊子之前，先来简单看下 IE 模型。

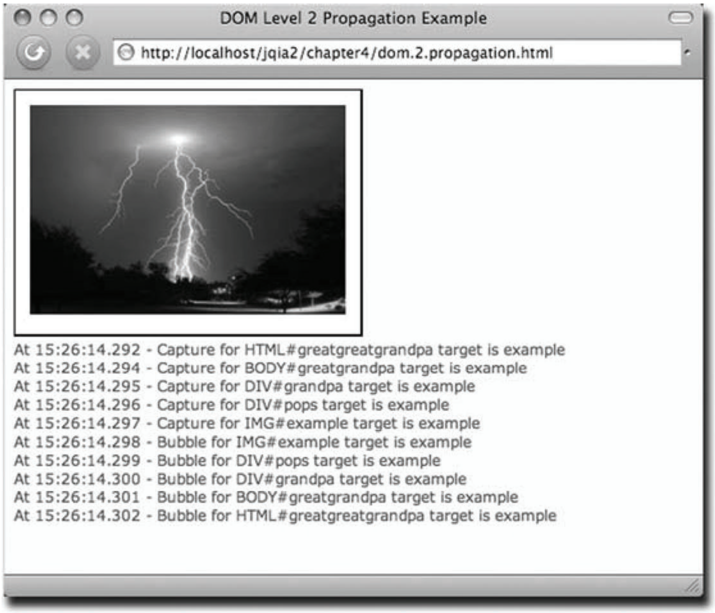


图 4-5　单击图片，使每个处理器都输出一条控制台消息，用于识别事件在捕获和冒泡阶段的传播路径

4.1.3　IE事件模型

IE 浏览器（包括 IE6、IE7 甚至最令人失望的 IE8）不支持 DOM 第 2 级事件模型。不过微软浏览器的这三个版本提供了专有的接口来模拟标准模型的冒泡阶段。

IE 模型为每个 DOM 元素定义了一个名为 `attachEvent()`的方法，而不是 `addEventListener()`。类似于标准模型，`attachEvent()`方法接受两个参数，如下所示：

```
attachEvent(eventName,handler)
```

第一个参数是字符串，用来指定需要附加的事件类型的名称。它不采用标准的事件名称，而是采用 DOM 第 0 级模型中相应的元素属性的名称——`onclick`、`onmouseover`、`onkeydown` 等。

第二个参数是需要创建的处理器函数，和在基本模型中一样，`Event` 实例必须通过 `window.event` 属性来获取。

这真是一团糟！即便使用相当独立于浏览器的 DOM 第 0 级模型时，在事件处理的各个阶段仍然面临众多混乱的依赖于浏览器的选择。而当使用更强大的 DOM 第 2 级模型或者 IE 模型时，我们甚至从一开始创建处理器时就不得不建立代码分支。

好了，jQuery 将尽其所能地将浏览器不一致的地方隐藏起来，从而使我们的开发更加简单。下面来瞧瞧 jQuery 是如何做到的！

4.2 jQuery 事件模型

虽然创建高交互性的应用需要高度依赖于事件处理，但是大规模地编写事件处理的代码以兼容不同浏览器的想法，足以让最无畏的页面开发者退缩。

我们可以把不一致性从页面代码中提取出来，将其隐藏在 API 中，但既然 jQuery 已经为我们完成了这一切，那为什么还要大费周折呢？

jQuery 事件模型的实现，也被非正式地称为 jQuery 事件模型，有如下特征：

- 为创建事件处理器提供了统一的方法；
- 允许为每个元素的每个事件类型创建多个处理器；
- 使用标准事件类型的名称，例如 `click` 或 `mouseover`；
- 使得 `Event` 实例可作为处理器的参数；
- 规范化 `Event` 实例中最常用的属性；
- 为取消事件和阻止默认行为提供统一的方法。

除了明显不支持捕获阶段之外，jQuery 事件模型的特征集和 DOM 第 2 级事件模型的特征集非常相似，都只使用一套 API 来支持标准兼容的浏览器和 IE 浏览器。由于捕获阶段缺少 IE 浏览器的支持，因此对于从不使用捕获阶段（甚至不知道它的存在）的大部分页面开发者来说，忽略此阶段应该不成问题。

真的这么简单？让我们来弄清楚。

4.2.1 使用jQuery绑定事件处理器

使用 jQuery 事件模型，可以利用 `bind()`方法在 DOM 元素上创建事件处理器。考虑如下简单的示例：

```
$('img').bind('click',function(event){alert('Hi there!');});
```

这个语句为页面上的每个图片绑定提供内联函数，作为单击事件处理器。`bind()`方法的完整语法如下所示。

下面实践一下 `bind()`方法。以代码清单 4-3 为例，把它从 DOM 第 2 级事件模型转换为 jQuery 事件模型，转换后的代码如代码清单 4-5 所示，也可以在文件 chapter4/jquery.events.html 中找到转换后的代码。

代码清单 4-5　不使用特定浏览器代码创建高级事件处理器

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Events Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript" src="../scripts/jqia2.support.js"></
     script>
    <script type="text/javascript">
```

```
      $(function(){                                    ◁┐
        $('#example')                                  ❶ 为图片绑定 3 个事件处理器
          .bind('click',function(event) {
            say('BOOM once!');
          })
          .bind('click',function(event) {
            say('BOOM twice!');
          })
          .bind('click',function(event) {
            say('BOOM three times!');
          });
      });
    </script>
  </head>

  <body>
    <img id="example" src="example.jpg"/>
  </body>
</html>
```

这段代码的修改只局限于就绪处理器的函数体中，虽然改动很小但是意义重大❶。我们创建由目标`<img>`元素组成的包装集，并对其应用 3 个 `bind()`方法（记住，jQuery 链允许在单个语句中应用多个方法），每个方法都在元素上创建一个单击（`click`）事件处理器。

方法语法：bind

bind(eventType, data, hanlder)

bind(eventMap)

创建一个函数，将其作为在匹配集中所有元素上指定的事件类型的事件处理器

参数

eventType	（字符串）为将要创建的处理器指定事件类型的名称。可以使用空格分隔的列表指定多个事件类型 通过在事件名称的后面添加圆点字符分隔的后缀，这些事件类型可以指定命名空间。本节的剩余部分会对其进行详细介绍
data	（对象）调用者提供的数据，用来附加到 Event 实例的 data 属性，以便为处理器函数所使用。如果省略，可以指定第二个参数为处理器函数
handler	（函数）将要创建为事件处理器的函数。当调用此函数时，会传入 Event 实例，并设置函数上下文（this）为冒泡阶段的当前元素
eventMap	（对象）一个 JavaScript 对象，允许在一次调用中创建多个事件类型的处理器。属性的名称指定事件类型（与 eventType 参数相同），属性的值提供处理器

返回值

包装集

在标准兼容浏览器中加载此页面，单击图片的结果如图 4-6 所示，这和我们在图 4-3 中看到的一模一样（除了 URL 和窗口标题），并不令人感到意外。

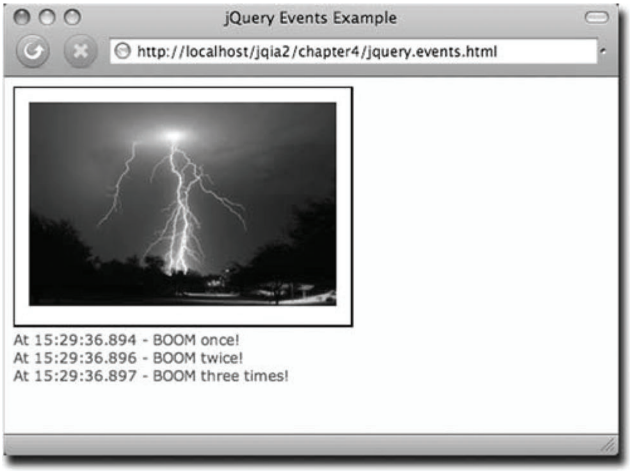


图 4-6　jQuery 事件模型允许指定多个事件处理器，就像 DOM 第 2 级事件模型那样

但或许更重要的是，即便用 IE 浏览器加载这段代码，网页也能正常工作，如图 4-7 所示。如果使用代码清单 4-3 中的代码，是做不到这一点的，除非向其中添加浏览器特定的测试和分支代码，以便为当前浏览器选择正确的事件模型。

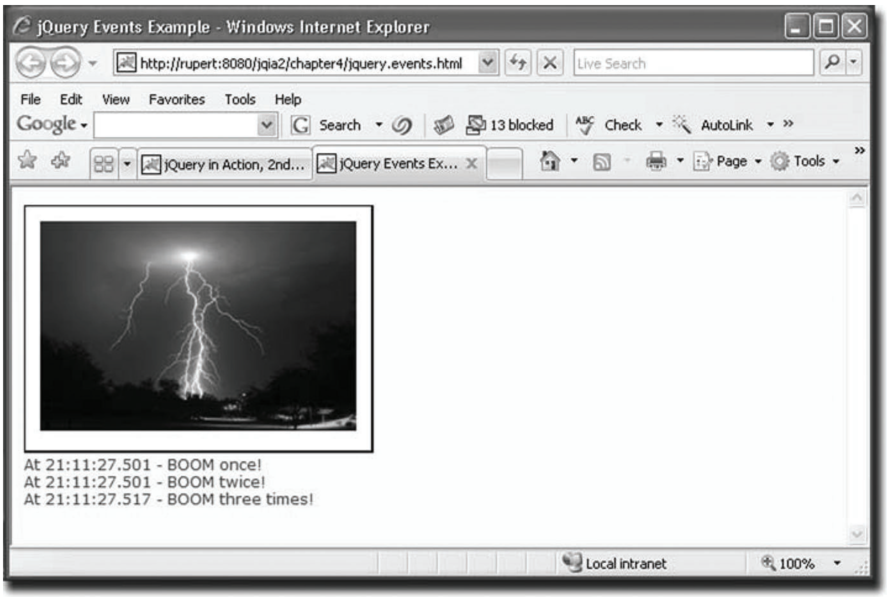


图 4-7　jQuery 事件模型允许使用统一的事件 API 来支持标准兼容的浏览器和 IE 浏览器

此刻，那些曾经与页面上堆积如山的浏览器特定事件处理代码做斗争的页面开发者，无疑会欢唱“快乐的日子又回来了”，并在办公室座椅上快乐地打转。谁又能责怪他们呢？

jQuery 还为事件处理提供了一个小巧的额外功能，就是通过指定命名空间来对事件处理器进行分组。和常规的命名空间不同（通过前缀指定命名空间），它通过为事件名称添加以圆点分隔的后缀来指定命名空间。事实上，如果你愿意，可以使用多个后缀将事件归入多个命名空间。

通过这种方式对事件绑定进行分组，随后就可以将它们看作一个单元以便对其进行操作。

例如，一个页面有两种模式：显示模式和编辑模式。在编辑模式下，事件监听器放置在页面的许多元素上，但是在显示模式下这些监听器都不适用了，因此当页面离开编辑模式时，就需要删除这些监听器。我们可以为编辑模式下的事件指定命名空间，代码如下所示：

```
$('#thing1').bind('click.editMode',someListener);
$('#thing2').bind('click.editMode',someOtherListener);
  ...
$('#thingN').bind('click.editMode',stillAnotherListener);
```

通过将这些绑定的事件分组到 editMode 命名空间，随后可以对它们进行整体操作。例如，当页面离开编辑模式需要删除所有的绑定时，可以使用如下代码轻松实现：

```
$('*').unbind('click.editMode');
```

这会为页面上的所有元素删除位于命名空间 editMode 中的所有 click 绑定（下一节会解释 unbind()方法）。

在结束对 bind()的学习之前，考虑另外一个示例：

```
$('.whatever').bind({
  click: function(event) { /* handle clicks */ },
  mouseenter: function(event) { /* handle mouseenters */ },
  mouseleave: function(event) { /* handle mouseleaves */ }
});
```

当需要为一个元素绑定多个事件类型的时候，在这种情况下可以通过调用单个 bind()来实现，如上所示。

除了 bind()方法，jQuery 还为创建特定的事件处理器提供了一些便捷方法。除了方法的名称不同之外，所有方法的语法都是相同的，为了节省一些空间，我们在如下的单个语法描述块中呈现所有的方法。

focusin 和 focusout 事件值得探讨。

不难想象这样的场景，我们需要集中控制获取和失去焦点事件。例如，假设需要跟踪表单里已被访问过的字段，如果能为表单创建单个处理器而不是为每个元素都创建一个处理器，这就会方便很多。但是我们做不到这一点。

从本质上来看，获取焦点和失去焦点的事件是不会沿着 DOM 树向上冒泡的。因此，建立在表单元素上的获取焦点事件处理器是不会被调用的。

这也是 focusin 和 focusout 之所以出现的原因。当可聚焦的元素获取或失去焦点时，会调用为元素上的这些事件创建的处理器，同时也会调用建立在其祖先元素上的任何此类处理器。

jQuery 还提供了 bind()方法的一个专用版本（名为 one()），用于创建一次性的事件处理器。一旦这个事件处理器执行了一次之后，就会被自动删除。它的语法类似于 bind()方法，如下所示：

方法语法：特定的事件绑定

***eventTypeName*(listener)**

创建一个指定的函数，将其作为与方法名称同名的事件类型的事件处理器。支持的方法如下：

blur	focusin	mousedown	mouseup
change	focusout	mouseenter	ready
click	keydown	mouseleave	resize
dblclick	keypress	mousemove	scroll
error	keyup	mouseout	select
focus	load	mouseover	submit
			unload

注意，当使用这些便捷方法时，不能指定存储在 event.data 属性中的 data 值

参数

listener　（函数）创建将要作为事件处理器的函数

返回值

包装集

方法语法：one

one(eventType, data, listener)

创建一个函数，并将其作为匹配集中所有元素指定事件类型的事件处理器。这个处理器一旦执行完毕，就会被自动删除

参数

eventType	（字符串）为将要创建的处理器指定事件类型的名称
data	（对象）调用者提供的数据，用来附加到 Event 实例中以便为处理器函数所使用。如果省略，则可以指定第二个参数为处理器函数
listener	（函数）将要被创建为一次性事件处理器的函数

返回值

包装集

这些方法为我们提供了将事件处理器绑定到所匹配元素的多种方式。而一旦绑定了一个处理器，最终我们可能需要删除它。

4.2.2　删除事件处理器

一般来说，只要创建了一个事件处理器，那么它在页面剩余的生命周期内就是有效的。但是一些特别的交互可能要求根据一定的标准删除处理器。考虑这个例子，一个用来呈现多个步骤的页面，一旦完成了某个步骤，这个步骤中的控件就需要还原到只读状态。

对于这种情况，在脚本控制下删除事件处理器比较好。我们已经看到 one()方法在第一次（也是仅有的一次）执行完毕后就会自动删除处理器，但是一般情况下，我们希望由自己控制删除事件处理器，为此 jQuery 提供了 unbind()方法。

unbind()的语法如下所示。

方法语法：unbind

unbind(eventType, listener)

unbind(event)

为包装集中的所有元素删除由可选的传入参数指定的事件处理器。如果没有提供参数，则会从元素上删除所有的监听器

参数

eventType	（字符串）如果提供，则只删除为指定事件类型创建的监听器
listener	（函数）如果提供，则找出将要删除的指定监听器
Event	（事件）删除触发 Event 实例描述的事件的监听器

返回值

包装集

这个方法可以用来从各个不同的粒度级别删除匹配集元素上的事件处理器。可以省略所有参数来删除所有的监听器，或者也可以只提供事件类型参数来删除指定类型的监听器。

可以通过提供初始时被创建为监听器的函数的引用，来删除特定的处理器。为了做到这一点，当一开始将函数绑定为事件监听器时，就必须保留函数的引用。出于这个原因，在最初创建作为监听器的函数（最终将被作为处理器删除）时，要么定义为顶级函数（以便通过其顶级变量名来引用），要么定义为通过其他方法保留的对该函数的引用。如果提供函数作为匿名的内联引用，那么在随后调用 unbind()时就不可能引用此函数。

在有些情况下，使用命名空间事件会带来不少方便，因为不需要保留监听器的单个引用就能解绑定特定命名空间下的所有事件。例如：

```
$('*').unbind('.fred');
```

这个语句将删除命名空间 fred 下的所有事件监听器。

目前为止，我们已经看到，jQuery 事件模型使得创建（删除也一样）事件处理器非常简单，而无需担心浏览器差异，但是如何编写事件处理器呢？

4.2.3　Event实例

当调用通过 bind()方法（或者任何相关的便捷方法）创建的事件处理器时，无论使用的是什么浏览器，Event 实例都会作为第一个参数传入函数，这样就不用担心 IE 浏览器下的 window.event 属性了。但是要如何处理 Event 实例中有差异的属性呢？

谢天谢地，不需要，因为老实说，jQuery 并没有真正把 Event 实例传入处理器。

吱！（针头在老唱片上拖动的声音。）

是的，我们一直对这个小细节闭口不谈，因为直到现在，它并没有什么影响。但是既然走到了这一步，我们将要考察处理器中的这个实例，就必须了解真相。

事实上，jQuery 定义了一个传入处理器的 jQuery.Event 类型的对象。但是我们的简化考虑也是可以原谅的，因为 jQuery 向这个对象复制了大部分的原始 Event 属性。如此一来，如果只想查找那些出现在 Event 中的属性，那么这个对象和原始的 Event 实例几乎是一样的。

但是这并非是该对象的重要之处——真正有价值的并且也是这个对象存在的原因就在于，它维护了一套规范化的值和方法，因此可以忽略 Event 实例的差异，独立于浏览器使用它们。

表 4-1 列出了以平台无关的方式可以安全访问的 jQuery.Event 属性和方法。

表4-1 独立于浏览器的jQuery.Event属性

名　称	描　述
属性	
altKey	当触发事件时，如果Alt键已被按下，则设置为true，否则为false。在大部分的Mac键盘中Alt键标记为Option
ctrlKey	当触发事件时，如果Ctrl键已被按下，则设置为true，否则为false
currentTarget①	冒泡阶段的当前元素。它和事件处理器中函数上下文对象是同一个对象
data	如果有值的话，在创建处理器时，将其作为第二个参数传入bind()方法
metaKey	当触发事件时，如果Meta键已被按下，则设置为true，否则为false。在PC上Meta键是Ctrl键，而在Mac上是Command键
pageX	对于鼠标事件，指定触发事件时光标相对于页面原点的水平坐标
pageY	对于鼠标事件，指定触发事件时光标相对于页面原点的垂直坐标
relatedTarget	对于鼠标事件，找出触发事件时光标离开或者进入的元素
screenX	对于鼠标事件，指定触发事件时光标相对于屏幕原点的水平坐标
screenY	对于鼠标事件，指定触发事件时光标相对于屏幕原点的垂直坐标
shiftKey	当触发事件时，如果Shift键已被按下，则设置为true，否则为false
result	从前面的事件处理器返回的最近的非undefined的值
target	找出触发事件的元素
timestamp	jQuery.Event实例创建时的时间戳，以毫秒为单位
type	为所有的事件指定触发的事件类型（例如click）。如果使用一个事件处理器来处理多个事件，那么这会非常有用

① 注意 currentTarget 和 target 的区别，比如在<body>上创建单击处理器，然后单击页面上的一个按钮，则处理器被调用时，event.target 是按钮元素，而 event.currentTarget 是<body>元素。

（续）

名　称	描　述
which	对于键盘事件，指定触发事件的按键的数字代码；对于鼠标事件，指定按下的是哪个按钮（1为左键、2为中键、3为右键）。应该使用which属性代替button属性，因为不能保证跨浏览器button属性的一致性
方法	
preventDefault()	阻止任意默认的语义动作（比如表单提交、链接重定向、复选框状态的改变等）发生
stopPropagation()	停止事件沿着DOM树向上进一步传播。当前目标元素上附加的事件不受影响。不仅支持浏览器定义的事件，而且支持自定义事件
stopImmediatePropagation()	停止所有事件的进一步传播，包括附加在当前目标元素上的事件
isDefaultPrevented()	如果已经在此实例上调用了preventDefault()方法，则返回true
isPropagationStopped()	如果已经在此实例上调用了stopPropagation()方法，则返回true
isImmediatePropagationStopped()	如果已经在此实例上调用了stopImmediatePropagation()方法，则返回true

使用 keyCode 属性在不同浏览器下处理非字母字符是不可靠的，认识到这一点很重要。例如，左箭头键的代码是 37，这在 keyup 和 keydown 事件中是可靠的，但是在 keypress 事件中，Safari 会为这些键返回非标准的结果。

可以在 keypress 事件中使用 which 属性获得一个可靠的、区分大小写的字符代码。在 keyup 和 keydown 事件中，只能获得一个不区分大小写的按键代码（因此 a 和 A 都返回 65），但是可以通过检查 shiftKey 属性来确定大小写。

还有，如果想停止事件的传播（而不是立即传播[①]），而且取消它的默认行为，那么可以让监听器函数返回 false。

这一切为 DOM 中现有的所有元素的事件处理器的创建和删除提供了细粒度的控制，但是对于那些还不存在的元素呢？

4.2.4　预先管理事件处理器

使用 bind() 和 unbind() 方法（以及很多便捷方法），可以很容易地控制要在 DOM 中的元素上创建哪些事件处理器。就绪处理器的方便就在于可以从头开始在 DOM 元素上创建处理器，这些 DOM 元素或者是通过页面上的 HTML 标记创建的，或者是在就绪处理器中创建的。

使用 jQuery 最主要的一个原因（正如我们在前一章看到的）是，它可以轻松地动态操作 DOM。当混合使用 Ajax 时（详见第 8 章），很有可能会在页面的生命周期内频繁引入 DOM 元素或删除

① 停止立即传播指的是阻止任何附加到目标元素上相同类型事件的执行。例如为一个按钮建立两个单击处理器，如果在第一个处理器内调用 event.stopImmediatePropagation()，则单击此按钮时，只有第一个处理器执行，第二个处理器不会执行。可以对比一下使用 event.stopPropagation() 的情况。

它们。在管理这些动态元素的事件处理器时，就绪处理器就起不到多大作用了，因为这些动态元素在就绪处理器执行时还不存在。

当使用 jQuery 操作 DOM 时，当然可以动态地管理这些事件处理器，但是如果能够将所有事件管理代码集中放在一个地方，岂不是更好？

1. 创建“live”事件处理

jQuery 提供了我们所期望的 `live()`方法，该方法允许预先为那些还不存在的元素创建事件处理器。

`live()`的语法如下。

方法语法：live

live(eventType, data, listener)

当指定类型的事件在元素（任何用来创建包装集的与选择器相匹配的元素）上发生时，会将传入的监听器作为处理器调用，而无论在调用 live 方法时这些元素是否已经存在

参数

eventType	（字符串）指定处理器将要调用的事件类型的名称。和 bind()不同，不能指定空格分隔的事件类型列表[①]
data	（对象）调用者提供的数据，用来附加到 Event 实例的 data 属性，以便为处理器函数所使用。如果省略，可以指定第二个参数为处理器函数
listener	（函数）将要作为事件处理器被调用的函数。当调用此函数时，会向此函数传入 Event 实例，并设置目标元素为函数上下文（this）

返回值

包装集

4

如果这个方法的语法让你想起 `bind()`方法的语法，那就对了。这个方法的定义和行为与 `bind()`很相似。不同之处是，当相应的事件发生时，该方法会为所有匹配选择器的元素触发此事件，甚至包括那些在调用 `live()`的时候还不存在的元素。

例如，可以这么写：

```
$('div.attendToMe').live(
  'click',
   function(event){ alert(this); }
);
```

在页面的整个生命周期内，单击任意拥有类 attendToMe 的<div>元素都会调用事件处理器并传入一个事件实例。而且前面的代码不需要放置于就绪处理器中，因为“live”事件不关心 DOM 是否已经存在。

① 这个说法不对，从 jQuery1.4 开始已经可以为 `live()`方法的第一个参数指定空格分隔的事件类型列表或者自定义事件类型了。

利用 live()方法，可以非常容易地在页面上的某个地方创建需要的事件处理器，而无需担心元素是否已经存在，或者什么时候创建元素。

但是使用 live()时必须要遵守一些注意事项。因为它和 bind()很相似，你可能会期望"live"事件完全按照与原生事件相同的方式工作，但是它们是有差异的，在某些页面中这种差异可能很重要，也可能无关紧要。

首先，要意识到"live"事件不是原生的"普通"事件。当类似于单击的事件发生时，它会沿着 DOM 元素向上传播（如前所述），并调用每一个已经创建的事件处理器。一旦事件到达用来创建包装集的上下文，也就是调用 live()的地方（通常是文档根节点[①]），上下文会在子节点中检查匹配"live"选择器的元素[②]。"live"事件处理器会在每个匹配的元素上触发，并且这个已触发的事件不能继续传播。

如果页面逻辑依赖于传播和传播顺序，live()可能不是最好的选择——特别是将"live"事件处理器和使用 bind()创建的原生事件处理器混合使用的时候。

其次，live()方法只能应用于选择器，不能应用于衍生而来的包装集。例如，下面两个表达式都是合法的：

```
$('img').live( ... )
$('img','#someParent').live( ... )
```

第一个表达式会影响所有的图片，第二个表达式只影响#someParent 创建的上下文中的所有图片。注意，当指定一个上下文时，它必须在调用 live()时已经存在。

但下面的表达式是不合法的：

```
$('img').closest('div').live( ... )
```

因为它是在其他的某个对象上调用 live()，而不是选择器。

虽然有这些限制，但是在使用动态元素的任何页面上使用 live()还是极其方便的，我们会在第 8 章看到在为页面引入 Ajax 时它是如何成为关键的要素的。在本章后面（4.3 节），我们将在一个综合示例中广泛使用 live()，这个示例使用第 3 章学的便捷的 DOM 操作方法来创建动态元素。

2. 删除"live"事件处理

live()创建的处理器可以使用 die()方法（令人讨厌的名字）来解绑定，它的语法和 unbind()非常相似。

除了允许以一种独立于浏览器的方式管理事件处理以外，jQuery 还提供了一套在脚本控制下触发事件处理器的方法。下面来看看这些方法。

---

① 从 jQuery1.4 开始，"live"事件不仅可以绑定到文档根节点，也可以绑定到一个 DOM 元素上下文。

② 浏览 jQuery 源代码，你会发现这个检查是通过$(event.target).closest('live selector')来查找匹配的元素的。

方法语法：`die`

`die(eventType, listener)`

删除由 `live()`创建的“live”事件处理器，并且阻止在将来创建的元素上调用处理器，这些元素是与调用 `live()`时使用的选择器相匹配的元素

参数

`eventType`	（字符串）如果提供，则只删除为指定事件类型创建的监听器
`listener`	（函数）如果提供，则找出将要删除的指定监听器

返回值

包装集

4.2.5 触发事件处理器

当浏览器或者用户活动触发了相关事件在 DOM 层次结构中传播时，事件处理器将被调用。但有时我们想在脚本控制下触发处理器的执行。可以将这样的事件处理器定义为顶级函数，以便通过名称调用它们。但是正如我们所看到的，将事件处理器定义为内联的匿名函数更加常用，并且还非常方便。此外，将事件处理器作为函数调用不会触发语义动作，也不会导致冒泡。

为了满足这一需求，jQuery 提供了在脚本控制下自动触发事件处理器的方法。这些方法中最常用的是 `trigger()`，它的语法如下所示。

方法语法：`trigger`

`trigger(eventType, data)`

在所有匹配元素上调用为传入的事件类型创建的处理器

参数

`eventType`	（字符串）指定将要调用处理器的事件类型的名称。这包括命名空间事件。可以在事件类型的后面追加感叹号（!），阻止命名空间事件的触发
`data`	（任意）将要作为第二个参数传入处理器的数据（在 event 实例之后）

返回值

包装集

`trigger()`方法（以及马上要介绍的其他便捷方法）尽力模拟事件的触发过程，包括在 DOM 层次结构中传播以及语义动作的执行。

每个处理器被调用时都会传入一个已填充的 `jQuery.Event` 实例。因为不存在真正的事件，因此用来报告事件相关值的属性（比如鼠标事件中光标的位置和键盘事件中按下的键）都是没有值的。`target` 属性被设置为与处理器绑定的匹配集元素的引用。

和实际的事件一样，触发事件的传播可以通过 `jQuery.Event` 实例的 `stopPropagation()` 方法，或者在任何调用的处理器中返回 `false` 来停止。

注意　传入 `trigger()`方法的数据参数和创建处理器时传入的数据参数是不一样的。后者可以通过 `jQuery.Event` 实例的 `data` 属性来调用，而传入 `trigger()`的值（后面介绍的 `triggerHandler()`也是同样的道理）将作为参数传入监听器。因此这两个值可以同时使用，彼此之间不会产生冲突。

对于只想触发处理器，而不需要事件传播和语义动作执行的情况，jQuery 提供了 `triggerHanler()`方法，它的行为看起来和 `trigger()`是一样的，不同之处是不会导致冒泡或者执行语义动作。此外，通过 `live` 绑定的事件不会被触发。

方法语法：`triggerHandler`

`triggerHandler(eventType, data)`

在所有匹配元素上调用为传入的事件类型创建的处理器，不会冒泡、不会执行语义动作，也不会触发“live”事件

参数

`eventType`	（字符串）指定将要调用处理器的事件类型的名称
`data`	（任意）将要作为第二个参数传入处理器的数据（在 `Event` 实例之后）

返回值

包装集

除了 `trigger()`和 `triggerHanlder()`方法，jQuery 还提供了便捷方法来触发大部分的事件类型。除了方法名称不同之外，这些方法的语法完全一样，如下所示。

方法语法：*`eventName`*

***`eventName`*`()`**

在所有匹配元素上调用为命名事件类型创建的处理器。支持的方法如下所示：

`blur`	`focusin`	`mousedown`	`resize`
`change`	`focusout`	`mousenter`	`scroll`
`click`	`keydown`	`mouseleave`	`select`
`dblclick`	`keypress`	`mousemove`	`submit`
`error`	`keyup`	`mouseout`	`unload`
`focus`	`load`	`mouseover`	

参数

无

返回值

包装集

除了绑定、解绑定以及触发事件处理器，jQuery 还提供了一些高级函数，使得页面上的事件处理尽可能简单。

4.2.6 其他事件相关的方法

交互应用页面经常会使用行为组合来实现交互样式。jQuery 提供了一些事件相关的便捷方法，使得在页面上使用这些交互行为更为简单。下面看看有哪些方法。

1. 起切换作用的监听器

第一个方法就是 `toggle()`，它为单击事件处理器创建了一个循环队列，并应用到每个随后发生的单击事件中。第一次单击事件时调用第一个注册的处理器，第二次单击时调用第二个处理器，第三次单击时调用第三个处理器，依此类推。当到达列表中的最后一个处理器时，第一个处理器就成为队列中下次将要调用的处理器。`toggle()`方法的语法如下所示

4

方法语法：toggle

`toggle(listener1, listener2, ...)`

将传入的函数创建为包装集中所有元素的单击事件处理器的循环列表。处理器将依次在随后的单击事件中被调用

参数

`listener`*N*　（函数）一个或者多个作为后续单击事件处理器的函数

返回值

包装集

这个便捷方法的一个常见用途是，根据元素被单击的次数（分奇数次单击和偶数次单击）切换元素的可用状态。为此，我们需要提供两个处理器：一个是奇数次单击处理器，另一个是偶数次单击处理器。

这个方法也可以使用两个或多个单击处理器来创建连续的行为。考虑如下示例。

假设有一个站点希望用户可以在 3 种尺寸下浏览图片：小号、中号和大号。这个交互过程可以通过一系列的单击事件来进行。单击图片会载入下一个更大尺寸的图片，直到最大尺寸，然后回到最小尺寸。

在图 4-8 中显示的是页面随时间推移的 3 个状态。单击一次图片时，它都会伸展到下一个更大的尺寸。到达最大尺寸后，如果再次单击图片，它就会回到最小尺寸。



图 4-8　jQuery 的 toggle()方法允许我们事先定义单击事件的一系列连续行为

这个页面的代码如代码清单 4-6 中所示，也可以在文件 chapter4/toggle.html 中找到。

代码清单 4-6　定义一系列连续的事件处理器

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery .toggle() Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css"/>
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript">
      $(function(){

        $('img[src*=small]').toggle(                         <-- 创建一系列连续的处理器
           function() {
             $(this).attr('src',
              $(this).attr('src').replace(/small/,'medium'));
           },
           function() {
             $(this).attr('src',
              $(this).attr('src').replace(/medium/,'large'));
           },
```

```
            function() {
              $(this).attr('src',
               $(this).attr('src').replace(/large/,'small'));
            }
         );

       });
     </script>
     <style type="text/css">
       img {
         cursor: pointer;
       }
     </style>
   </head>

   <body>

     <div>Click on the image to change its size.</div>
     <div>
       <img src="hibiscus.small.jpg" alt="Hibiscus"/>
     </div>

   </body>
 </html>
```


注意　如果你和本书作者一样并且注意观察事物的名称，可能会想为什么这个方法被命名为 toggle()，并且只有在创建了两个处理器时这个名字才显得有意义。原因是在 jQuery 的早期版本中，这个方法仅限于指定两个处理器，后来被扩展为可以接受任意数量的处理器。为了向后兼容这个名字仍然保持不变。

另一个在交互应用中经常遇到的多事件场景涉及鼠标进入和移出元素。

2. 在元素上悬停鼠标

通知我们鼠标光标在某时进入某区域或者离开某区域的事件，对于在页面上创建呈现给用户的很多常见用户界面元素来说，这是必不可少的。在这些元素类型中，作为导航系统的层叠菜单是一个常见示例。

mouseover 和 mouseout 事件类型的一个恼人行为经常会妨碍我们轻松创建这类元素：当鼠标进入某区域及其子节点时会触发 mouseout 事件。考虑图 4-9 的显示（可从文件 chapter4/hover.html 中获取）。

这个页面显示了两组相同（除了名称以外）的区域：每组都包括一个外部区域和一个内部区域。在浏览器中加载这个页面并阅读本节下面的内容。

对于页面顶部的那组矩形区域，就绪处理器中创建 mouseover 和 mouseout 事件处理器的脚本如下：

```
$('#outer1').bind('mouseover mouseout',report);
$('#inner1').bind('mouseover mouseout',report);
```

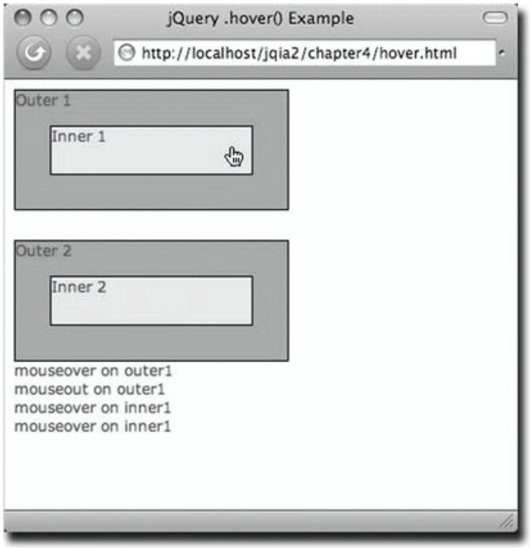


图 4-9　这个页面说明了当鼠标光标移入某区域及其子节点时所触发的鼠标事件

这段代码创建了名为 report 的函数作为 mouseover 和 mouseout 的事件处理器。

report()函数的定义如下所示：

```
function report(event) {
  say(event.type+' on ' + event.target.id);
}
```

这个监听器只是把一个包含触发事件名称的<div>元素，添加到在页面底部定义的名为 console 的<div>元素里，以便让我们观察事件触发的时间。

现在，移动鼠标光标进入标记为 Outer 1 的区域（小心不要进入 Inner 1）。我们将看到（观察页面底部的输出）触发了一个 mouseover 事件。将光标移出此区域。不出所料，触发了一个 mouseout 事件。

刷新页面以便清空控制台重新开始。

现在，移动鼠标指针进入 Outer 1（注意触发的事件），但是这次继续向内移动直到指针进入 Inner 1。在鼠标进入 Inner 1 的时候，Outer 1 的 mouseout 事件被触发了。如果在 Outer 1 和 Inner 1 的边界来回晃动指针，将会看到一连串的 mouseout 和 mouseover 事件。这就是定义的行为，尽管这相当不直观。虽然指针仍然留在 Outer 1 的范围之内，但是当指针进入内部的元素时，事件模型会认为这个转变是离开外部区域。

不论是否预料到，我们并不总是需要那样的行为。我们经常会希望在指针离开外部区域边界时得到提示，而不关心指针是否位于内部区域。

幸运的是，一些主要的浏览器支持一对非标准的鼠标事件（mouseenter 和 mouseleave），它们最早由微软的 IE 浏览器引入。这对事件的行为稍微直观一些，当鼠标从一个元素移动到其子孙元素时不会触发 mouseleave 事件。对于不支持这些事件的浏览器，jQuery 模拟它们的实现，

以便在所有的浏览器中它们的工作方式相同。

使用 jQuery 可以用如下代码为这组事件创建处理器：

```
$(element).mouseenter(function1).mouseleave(function2);
```

jQuery 也提供了一个更加简单的单个方法：hover()。语法如下所示。

方法语法：hover

hover(enterHanlder, leaveHandler)

hover(handler)

为匹配元素创建 mouseenter 和 mouseleave 事件处理器。这些处理器只会在鼠标进入或者离开元素覆盖的区域时被触发，而忽略移动到子元素的转变

参数

enterHandler	（函数）将要成为 mouseenter 处理器的函数
leaveHandler	（函数）将要成为 mouseleave 处理器的函数
handler	（函数）mouseenter 和 mouseleave 事件发生时都会调用的单个处理器

返回值

包装集

在 hover.html 示例页面中，使用如下脚本为第二组区域（Outer 2 及其子元素 Inner 2）创建鼠标事件处理器：

```
$('#outer2').hover(report);
```

和第一组区域一样，report()函数被创建为 Outer 2 的 mouseenter 和 mouseleave 事件处理器。但是和第一组不同，当我们在 Outer 2 和 Inner 2 的边界上移动鼠标指针时，这些处理器（Outer 2 的处理器）都不会被调用。对于有些情况，当我们不希望父元素处理器响应鼠标指针移动到子元素的事件时，这会非常有用。

下面使用迄今为止学到的所有事件处理工具来研究一个示例页面。这个示例不仅使用了这些工具，而且也用到了前面几章学到的其他 jQuery 技术。

4.3　充分利用（更多的）事件

既然我们已经探讨了 jQuery 如何清理跨浏览器不同事件模型处理的混乱并重建了秩序，下面就来研究一个示例页面，将迄今为止学到的知识都用上。这个示例不仅用到了事件，而且也用到了前面几章探讨的一些 jQuery 技术，包括一些重量级的 jQuery 方法链。为了更好地理解这个综合示例，我们假设自己是视频发烧友，所收藏的成千上万张 DVD 已经成为一个巨大的问题。不仅如何组织这些 DVD 是个问题，就是迅速找出某张 DVD 也变得很困难，而且所有的这些 DVD 都放在盒子里，存储也成了问题——它们占用了太多空间，如果这个问题解决不了的话，我们只能将它们扔出房子。

一个保存一百张 DVD 的活页夹，与存放在盒子里的相当数量的 DVD 相比，占用的空间更少，因此可以通过购买 DVD 活页夹来解决存储问题。虽然这样可以避免我们睡在公园的长椅上[①]，但是如何组织 DVD 光盘仍然是一个问题。如何找到一张需要的 DVD，而无需手工翻动每一个活页夹直到找到需要的那张呢？

为了快速地定位某张特定的 DVD 光盘，我们不能用类似于按字母排序的方法来组织光盘顺序。那样做意味着每购买一个新的 DVD，就需要移动也许几十个活页夹中的所有光盘来保持已收藏光盘的顺序。想象一下在购买 *Abbott and Costello Go to Mars*[②]之后的工作量吧！

幸好我们有电脑，知道如何编写 Web 应用程序，并且还有 jQuery！因此，我们将编写一个 DVD 数据库程序来帮助跟踪所拥有的 DVD 以及它们的存放位置，以解决这个问题。

下面开始工作吧！

4.3.1　过滤大的数据集合

我们的 DVD 数据库程序面临的问题与很多其他程序遇到的一样，即究竟应该采用基于 Web 的形式还是其他形式？如何让用户（在这个案例中是我们自己）迅速地找到他们寻求的信息？

可以采用低技术含量的方式，仅仅显示一个包含所有标题的排序列表，但如果条目数量很多，滚动列表仍然很痛苦。此外，我们想要学习如何正确地实现这一目标，以便可以将学到的知识应用到真正面向客户的应用程序中。

因此无捷径可循！

显然，设计整个应用程序远远超出了本章的范围，因此我们将专注于开发一个控制面板，它将允许我们指定过滤器，用于调整数据库搜索返回的标题列表。

当然，我们希望能够过滤 DVD 的标题。但是我们也希望能够添加其他的过滤条件，根据电影发布的年份、分类、放置光盘的活页夹乃至是否观看过这部电影（这将有助于回答常见的问题，“今晚我应该看什么？”）来查找所需的 DVD。

你最初的反应可能是想知道这没什么大不了的。毕竟，我们可以放置多个字段来完成检索，对吧？

嗯，等一下。类似于标题的单个字段是没问题的，例如查找所有标题中包含“creature”的电影。但是，如果我们想搜索“creature”或者“monster”，该怎么办呢？或者只在 1957 年或 1972 年发布的电影呢？

为了给指定过滤器提供一个健壮的接口，我们需要允许用户指定多个过滤器，用来过滤 DVD 相同或不同的属性，而不是试图猜测将需要多少过滤器。我们对此完全有把握并可以按需创建这些过滤器。

我们将借用 Apple 用户界面手册中的一个页面，并模仿很多 Apple 应用程序中指定过滤器的

① 这是夸张的说法，指 DVD 已经占据了整个房子，不得不露宿在外面。

② 1953 年出品的科幻电影，中文译名为《两傻飞渡海神星》。

方式来创建界面。（如果你是 iTunes 用户，可以参考一下 Smart Playlists 是如何创建的。）

每一个过滤器占据一行，通过下拉列表（单选的`<select>`元素）来指定将要过滤的字段。根据该字段的类型（字符串、日期、数字，甚至是布尔型）在同一行中显示合适的控件以捕捉过滤器的相关信息。

用户可以依其所好添加任意数量的过滤器，或者删除之前指定的过滤器。

一张图片胜过千言万语，图 4-10a 到图 4-10c 显示了随时间推进的界面显示。这些图片显示了我们创建的过滤面板：(a)初始显示，(b)指定一个过滤器后的显示以及(c)指定很多过滤器后的显示。



图 4-10a　初始显示单个未配置的过滤器



图 4-10b　选中一个过滤器类型后，添加其限定符控件



图 4-10c　用户可以添加多个需要的过滤器

观察图 4-10a 到图 4-10c 的交互过程，就可以发现有很多元素是动态创建的。接下来将花一些时间来讨论如何着手去完成这个示例。

4.3.2　通过模板复制创建元素

很容易看到，为了实现这个过滤控制面板，将需要创建相当多的元素以响应各种事件。例如，当用户单击 Add Filter 按钮时，将需要创建一个过滤器条目，并且当一个特定的字段被选中时，将创建用来限定过滤器的新控件。

没问题！在前一章中，我们已经看到使用 jQuery 提供的`$()`函数可以很容易地以动态的方式创建元素。虽然我们会在示例中使用这种方法，但还将探索一些更高级的替代方法。

当动态地创建大量元素时，创建这些元素的所有代码可能略显笨拙，保持它们之间的关联可能有点难以维护，即便在 jQuery 的帮助下依然如此。（没有 jQuery 的帮助，这将完全是一个噩梦！）如果能够使用 HTML 为复杂的标记创建一个“蓝图”，每当我们需要这个蓝图的实例时就可以复制出来，这个办法很棒，对吧。

太棒了！jQuery 的 `clone()`方法刚好为我们提供了这种能力。

我们将要采取的做法是，创建几组“模板”标记用来呈现希望复制的 HTML 片段，并在需要的时候使用 `clone()`方法创建当前模板的实例。我们不希望这些模板对最终用户可见，因此在页面的结尾将这些模板放置于使用 CSS 隐藏的一个`<div>`元素中。

作为示例，请考虑下由“X”按钮和下拉列表组合获取的可过滤字段。在每次用户单击 Add Filter 按钮时，我们都需要创建这个组合的一个实例。利用 jQuery 创建这样的一个按钮和`<select>`元素及其`<option>`子元素会使代码过长，尽管编写和维护起来可能不会很麻烦。但是可以很容易联想到，如果事情更加复杂代码就会很快变得臃肿起来。

使用模板技术，将按钮和下拉列表的模板标记放在用来隐藏所有模板的<div>父元素中，可以创建如下的标记：

```
<!-- hidden templates -->
<div id="templates">                                    ❶ 包含并隐藏所有的模板
  <div class="template filterChooser">                  ❷ 定义 filterChooser 模板
    <button type="button" class="filterRemover" title="Remove this
     filter">X</button>

    <select name="filter" class="filterChooser" title="Select a property to
     filter">
      <option value="" data-filter-type="" selected="selected">
        -- choose a filter --</option>
      <option value="title" data-filter-type="stringMatch">DVD Title</option>
      <option value="category" data-filter-type="stringMatch">Category
      </option>
      <option value="binder" data-filter-type="numberRange">Binder</option>
      <option value="release" data-filter-type="dateRange">Release Date
      </option>
      <option value="viewed" data-filter-type="boolean">Viewed?</option>
    </select>
  </div>

  <!-- more templates go here -->

</div>
```

将 id 值为 templates 的外部<div>作为所有模板的容器，并且设置其 CSS 的 display 规则为 none，以阻止它在浏览器中显示出来❶。

在这个容器中定义了另一个<div>，赋予它类名 template 和 filterChooser❷。通常使用 template 类来找出模板，使用 filterChooser 类找出这个特殊的模板类型。很快我们将看到如何在代码中使用这些类——记住，类不仅仅用于 CSS！

另外请注意，<select>中的每个<option>都被赋予了自定义特性：data-filter-type。这个值将用于确定为选中的过滤字段应用什么类型的过滤控件。

基于已确定的过滤器类型，我们将选择适合这种过滤器类型的限定控件来填充当前过滤项所在“行”的剩余控件。

例如，如果过滤器类型为 stringMatch，需要显示一个用户可以向其中输入搜索词的文本输入框，以及一个下拉列表，该列表提供了多个可选项供用户选择如何应用搜索词。为这组控件创建的模板如下所示：

```
<div class="template stringMatch">
  <select name="stringMatchType">
    <option value="*">contains</option>
    <option value="^">starts with</option>
    <option value="$">ends with</option>
    <option value="=">is exactly</option>
  </select>
  <input type="text" name="term"/>
</div>
```

我们再次使用 template 类来识别一个模板元素，并且使用类 stringMatch 标记此元素。

这样做的目的是使这个类与过滤器字段选择下拉列表中的 data-filter-type 值一致。

无论何时何地，我们希望都能使用现有的 jQuery 知识来轻松复制这些模板。比方说希望在某个元素的末尾添加模板实例，而这个元素可以通过变量名 whatever 来引用。可以这样编写代码：

```
$('div.template.filterChooser').children().clone().appendTo(whatever);
```

在这个语句中，选择要复制的模板容器（使用放置于模板标记中的类），选择模板容器的子元素（不需要复制<div>，只需要复制它的内容），复制这些子节点，然后将它们添加到由 whatever 识别的元素内容的末尾。

现在明白为什么我一直强调 jQuery 方法链强大的原因了吧？

查看 filterChooser 下拉列表的选项就可以看到已经定义了很多其他的过滤器类型：numberRange、dateRange 以及 boolean。因此我们也为这些过滤器类型定义了限定控件模板，代码如下所示：

```
<div class="template numberRange">
  <input type="text" name="numberRange1" class="numeric"/> <span>through
     </span>
  <input type="text" name="numberRange2" class="numeric"/>
</div>

<div class="template dateRange">
  <input type="text" name="dateRange1" class="dateValue"/>

  <span>through</span>
  <input type="text" name="dateRange2" class="dateValue"/>
</div>

<div class="template boolean">
  <input type="radio" name="booleanFilter" value="true" checked="checked"/>
     <span>Yes</span>
  <input type="radio" name="booleanFilter" value="false"/> <span>No</span>
</div>
```

好，现在已经确定了复制策略，下面就来看下基础标记。

4.3.3　建立主体标记

如果回顾图 4-10a，就会发现 DVD 搜索页面的初始显示非常简单：一些标题、第一个过滤器实例以及两个按钮。来看下实现这个页面的 HTML 标记：

```
<div id="pageContent">

  <h1>DVD Ambassador</h1>
  <h2>Disc Locator</h2>

  <form id="filtersForm" action="/fetchFilteredResults" method="post">

    <fieldset id="filtersPane">
      <legend>Filters</legend>
      <div id="filterPane"></div>        ❶ 过滤器实例的容器
      <div class="buttonBar">
        <button type="button" id="addFilterButton">Add Filter</button>
        <button type="submit" id="applyFilterButton">Apply Filters</button>
```

```
        </div>
      </fieldset>

      <div id="resultsPane">
        <span class="none">No results displayed</span>
      </div>

    </form>

  </div>
```

❷ 搜索结果的容器

这个标记里没有太令人奇怪的地方——有吗？例如，初始的过滤器下拉列表的标记在哪里？我们已经建立了一个放置过滤器的容器❶，但是它初始时是空的。这是为什么？

嗯，我们需要能够动态地填充新的过滤器（很快就会介绍到），因此为什么需要在两个地方做这个事情呢？随后你将看到，可以利用动态代码来初始填充第一个过滤器，因此没有必要在静态标记中显式地创建它。

应该指出的另一件事情是，我们预留了用来接收结果❷的一个容器（如何获取结果超出了本章的范围）。我们把这些结果放置在表单中，因此结果本身可以包含表单控件（为了排序、分页等）。

好的。我们已经有了非常简单的主体 HTML 布局，而且还有一些隐藏的模板，可以通过复制它们来迅速地产生新的元素。最后看下如何编写代码来为页面添加行为！

4.3.4 添加新的过滤器

单击 Add Filter 按钮后，就需要添加一个新的过滤器到`<div>`元素中，这个用来接收过滤器的`<div>`元素是通过 `id` 为 `filterPane` 来指定的。

还记得吧，使用 jQuery 来创建事件处理器简直是小菜一碟，为 Add Filter 按钮添加一个单击处理器应该是一件简单的事情。但是别急！我们忘了考虑一些事情！

我们已经看到当用户添加过滤器时如何复制表单控件，已经有一个很好的策略用来很容易地创建这些控件的多个实例。但是最终必须将这些值提交到服务器，以便服务器可以在数据库中查找筛选的结果。如果只是一遍又一遍地为控件复制相同的 name 特性，服务器只会得到一堆乱七八糟的东西，而不知道哪个限定词属于哪个过滤器！

为了帮助编写服务器端代码（无论如何这都是一个自己编写代码的好机会），我们将为每个过滤项的 name 特性追加一个唯一的后缀。我们将保持简洁并使用计数器，将在第一组过滤器控件的名称后面追加“.1”，而第二组追加的是“.2”，以此类推。这样，当这些过滤器控件作为请求的一部分到达服务器时，服务器端代码可以根据后缀对它们进行分组。

注意 这个示例代码之所以选择“ .n ”作为后缀格式，是因为它简单而且从概念上代表了后缀试图表达的意思（对参数数据进行分组）。根据使用的服务器端代码的不同，你可能希望选择其他替代的后缀格式，结合可用的数据绑定机制会工作得更好。例如，“ .n ”格式与使用基于属性的 POJO 绑定机制的 Java 后端配合的不是太好（这种情况下，“ [n] ”格式可能会更合适）。

为了跟踪已经添加的过滤器个数（这样可以将此计数作为随后的过滤器名称的后缀），我们将创建一个初始化为 0 的全局变量，如下所示：

```
var filterCount = 0;
```

“全局变量？我认为它们是有害的。”我听到你这么说。

如果使用不当，全局变量可能会是一个问题。但在这种情况下，全局变量的确代表一个页面范围内的全局值，并且它绝对不会制造任何冲突，因为页面的各个地方都将会以一致的方式访问这个唯一值。

创建这个变量之后，将下面代码添加到就绪处理器中，为 Add Filter 按钮建立单击处理器（记住，在知道 DOM 元素被创建之后才能引用它们）：

```
$('#addFilterButton').click(function(){          ❶ 建立单击处理器
  var filterItem = $('<div>')                     ❷ 创建过滤器项目块
    .addClass('filterItem')
    .appendTo('#filterPane')
    .data('suffix','.' + (filterCount++));
  $('div.template.filterChooser')                 ❸ 复制过滤器下拉列表模板
      .children().clone().appendTo(filterItem)
      .trigger('adjustName');                     ❹ 触发自定义事件
});
```

虽然这个复合语句初看起来很复杂，但它却使用很少的代码完成了大量的功能。下面来分解各个步骤。

这段代码所做的第一件事情是使用 jQuery 的 `click()` 方法在 Add Filter 按钮上建立单击处理器❶。在单击按钮时将调用传入这个方法的函数，此时所有有趣的事情就会在函数内发生了。

因为每次单击 Add Filter 按钮都会添加一个新的过滤器，所以我们为这个过滤器创建了一个新的容器❷来放置它。为容器添加的类 `filterItem` 不仅为了 CSS 样式，也为了在以后的代码中能够找到这些元素。创建容器元素后，将其追加到 `id` 为 `filterPane` 的主过滤器容器中。

我们也需要记录下要添加到控件名称的后缀，这些控件将会放置于 `filterItem` 容器中。在需要调整控件名称的时候会用到这个后缀，这是一个不适合保存为全局变量值的好例子。每个过滤容器（类 `filterItem`）将拥有自己的后缀，因此用全局变量记录这个值将需要某种复杂的数组或者表结构，以便确保不同的值不会相互覆盖。

相反，使用非常便捷的 jQuery 的 `data()` 方法将后缀值记录在元素上可以避免这种混乱。之后，当需要知道为控件使用什么后缀时，我们只需查看控件容器上的这个数据值，而不用担心与记录在其他容器上的值相混淆。

代码片段：

```
.data('suffix','.' + (filterCount++))
```

使用 `filterCount` 变量的当前值格式化后缀值，然后将 `filterCount` 加 1。使用名称 `suffix` 将这个后缀值附加到 `filterItem` 容器上，以后每当我们需要它的时候就可以通过这个名称来获取。

单击处理器的最后一条语句❸使用我们在前一节讨论过的方法，来复制已创建的包含过滤下拉列表的模板。你可能会认为此时工作已经完成了，但是在复制和追加之后执行❹了如下的代码片段：

```
.trigger('adjustName')
```

trigger()方法用来触发一个名为 adjustName 的事件处理器。

adjustName?

如果查阅规范，你会发现无法找到这个事件！它是一个只在本页面中定义的非标准事件。这段代码所做的事情是触发了一个*自定义事件*。

自定义事件是一个非常有用的概念——我们可以将代码作为自定义事件处理器附加在元素上，并通过触发事件来促使其执行。与直接调用代码相比，这种方法的优点是可以预先注册自定义处理器，并通过触发事件来促使任何已经注册的处理器执行，而无需知道这些处理器是在哪里建立的。

模式提醒

jQuery 中自定义事件能力是观察者模式的一个有限制的示例，有时也被称为发布/订阅模式。通过在元素上创建特定事件的处理器来订阅元素的该事件，然后当该事件发布（被触发）时，在事件层次结构中，任何已订阅的元素都会自动调用它们的处理器。这样就可以通过减少必要的耦合来大大降低代码的复杂度。

之所以称其为观察者模式的有限制的示例，是因为订阅者局限于发布者祖先层次结构中的元素（而不是 DOM 中的任意元素）。

因此上面的代码将会触发自定义事件，然而我们需要定义该事件的处理器，所以在就绪处理器中也创建了如下代码来调整过滤器的控件名称：

```
$('.filterItem [name]').live('adjustName',function(){
  var suffix = $(this).closest('.filterItem').data('suffix');
  if (/(\w)+\.(\d)+$/.test($(this).attr('name'))) return;
  $(this).attr('name',$(this).attr('name')+suffix);
});
```

这里我们看到，live()方法可以用来预先建立事件处理器。每当添加或者删除一个过滤器时，将会添加或者删除拥有 name 特性的输入元素，因此引入 live()来在必要的时候自动建立或者删除处理器。这样一来，我们只需设置一次，jQuery 就能处理匹配.filterItem [name]选择器的项被创建或者销毁时的细节，是不是很简单？

我们指定自定义事件的名称为 adjustName，并且提供了每当自定义事件触发时要执行的处理器。（因为 adjustName 事件是一个自定义事件，所以不可能被用户活动触发，也就是说，不可能以类似单击处理器的方式工作。）

在处理器中，我们获得了记录在 filterItem 容器上的后缀。记住，在处理器中，this 指向触发事件的元素，也就是拥有 name 特性的元素。closest()方法可以快速地找到父容器，并在此元素上找到后缀值。

我们不希望调整已经修改过一次的元素名称，因此使用正则表达式来测试名称是否已经有附加的后缀。如果有，就从处理器返回。

如果尚未调整元素的名称，则使用 attr()方法获取原始名称，并将调整后的元素名称返回给元素。

在这一点上值得深思的是，如何将这段代码作为自定义事件实现，以及使用 live()在页面代码中建立非常松散的耦合。这使得我们不用担心要显式地调用调整名称的代码，或者当需要应用自定义处理器时要在代码的各个地方显式地建立这些处理器。这不仅保持了代码的简洁，而且增加了代码的灵活性。

在浏览器中加载此页面，测试 Add Filter 按钮的行为。注意，每次单击 Add Filter 按钮时，新的过滤器是如何被添加到页面中的。如果使用 JavaScript 调试器（Firefox 中的 Firebug 能够胜任这个工作）来检查 DOM，你将会看到每个<select>元素的名称都被附加了过滤器特有的后缀，就像预期的那样。

但是我们的工作还没有结束，下拉列表还不能指定将要过滤的字段。当用户选择其中的一项时，需要使用与该字段的过滤类型相对应的限定控件来填充过滤容器。

4.3.5　添加限定控件

每当选中过滤器下拉列表中的一项时，就需要使用适合该过滤器的控件来填充它。我们已经通过创建标记模板来简化了操作，只需在确定了合适的模板之后复制它就行了，但还需要在下拉列表的值改变时完成一些其他的日常任务。

下面看看为下拉列表建立改变处理器时需要完成哪些事情：

```
$('select.filterChooser').live('change',function(){          ❶ 建立改变处理器
  var filterType = $(':selected',this).attr('data-filter-type');
  var filterItem = $(this).closest('.filterItem');
  $('.qualifier',filterItem).remove();                        ❷ 删除任何旧的控件
  $('div.template.'+filterType)
      .children().clone().addClass('qualifier')               ❸ 复制合适的控件
      .appendTo(filterItem)
      .trigger('adjustName');
  $('option[value=""]',this).remove();                        ❹ 删除过时的选项
});
```

我们再次利用 jQuery 的 live()方法预先建立了处理器，这个处理器会在适当的时间点自动执行，而无需采取进一步的行动。这次，我们为任何将要生成的过滤器下拉列表预先建立了改变处理器❶。

提示　使用 live()指定改变事件是 jQuery1.4 新增的（也是很受欢迎的）功能。

当触发改变处理器时，首先收集几条信息：记录在自定义 data-filter-type 特性上的过滤器类型以及父级过滤容器。

一旦得到了这些值，我们就需要删除任何可能已经存在于容器中的过滤器限定控件❷。毕竟，用户可以多次改变选中的字段值，我们也不希望随着程序的运行不断地添加越来越多的控件！当所有适合的元素被创建时，我们将为它们添加 `qualifier` 类（在下一个语句中），以便轻松地选择和删除它们。

一旦确信清除了旧的元素，我们就使用从 `data-filter-type` 特性获取的值来复制模板以创建正确的限定控件集合❸。为每一个已创建的元素添加 `qualifier` 类名的目的是为了便于选择（正如在前一个语句中看到的那样）。另外请注意我们是如何再次触发 `adjustName` 自定义事件，来自动触发用来调整新创建控件的 `name` 特性的处理器的。

最后，我们希望删除过滤器下拉列表中的“choose a filter”`<option>`元素❹，因为一旦用户选中了一个特定的字段，再次选择那个条目就没有任何意义了。我们也可以忽略用户选中该选项时触发的改变事件，但是防止用户做一些没意义事情的最好方式，是在初始时不要让他们做这件事情！

再次，在浏览器中打开示例页面。尝试添加多个过滤器，并改变它们的选中项。注意限定控件是如何总是匹配选中字段的。如果可以在调试器中查看 DOM，观察 `name` 特性是如何被扩增的。

现在来看下删除按钮。

4.3.6 删除不需要的过滤器和其他任务

我们赋予了用户改变字段的能力，该字段将会应用任意的过滤器，同时也提供了删除按钮（标记为“X”），来完全删除过滤器。

此时，你应该已经意识到，使用已掌握的工具来完成这个任务几乎是轻而易举的事情。当按钮被单击时，所需要做的就是找出最近的父级过滤容器并删除它。

```
$('button.filterRemover').live('click',function(){
  $(this).closest('div.filterItem').remove();
});
```

是的，就是这么简单。

但还有一些其他事情……

你可能会回想起当页面第一次加载时，显示了一个初始的过滤器，尽管没有在标记中包含它。我们可以在页面加载后，模拟一次 Add Filter 按钮的单击行为来实现这个功能。

因此，在就绪处理器中添加以下代码：

```
$('#addFilterButton').click();
```

这会调用 Add Filter 按钮处理器，就好像用户单击了它一样。

最后一件事情。尽管处理表单到服务器的提交超出了本示例的讨论范围，不过还是给你透露一些在未来几章中将会出现的诱人的内容。

下面继续操作，单击 Apply Filters 按钮，你可能已经注意到这是一个表单的提交按钮。但是页面并不像预料的那样重新加载，而是将结果显示在 `id` 为 `resultsPane` 的`<div>`元素中。

这是因为我们在自己的处理器中取消了表单提交，取而代之的是，使用表单内容发起了一次

Ajax 请求并将结果显示在 resultsPane 容器中，从而破坏了表单的提交动作。我们将在第 8 章看到更多有关 jQuery 如何简化 Ajax 操作的内容，但是你可能会惊讶地看到（特别是如果你曾经做过一些跨浏览器的 Ajax 编程），我们只用一行代码就能完成 Ajax 请求：

```
$('#filtersForm').submit(function(){
  $('#resultsPane').load('applyFilters',$('#filtersForm').serializeArray());
  return false;
});
```

如果留意这些显示结果，你可能已经注意到无论指定什么过滤器结果都一样。显然，这里没有真正的数据库支撑这个示例，所以它实际上只是返回一个硬编码的 HTML 页面。但是不难想象，传递给 jQuery 的 Ajax 方法 load()的 URL 可以指向能够返回真实结果的资源，比如动态的 PHP、Java 小程序或者 Rails 资源。

这样就完成了这个页面，至少完成了本章所期望达到的目标，但是我们都知道……

4.3.7　总是有改进的余地

要使这个过滤器表单达到产品级的质量，还是有很多改进余地的。

下面列出了一些额外的功能，它们或者是表单被视为完整之前需要实现的，或者只是锦上添花的功能。你能够使用目前已获得的知识来实现这些附加功能吗？

- 表单缺少数据验证。例如，活页夹范围的限定字段应该是数字值，但是为了防止用户输入无效值我们什么也没做。同样的问题也存在于日期字段中。

  我们可以撒手不管让服务器端代码处理它——毕竟，服务器端无论如何都要进行数据验证。但是这会导致不愉快的用户体验，我提到过，处理错误的最好方式是在一开始就阻止其发生。

  因为解决方案涉及检查 Event 实例（目前还不包含在本示例中），所以将给出不允许输入任何字符而只能输入十进制数字到数字字段的代码。有了在本章中学到的知识，你应该很熟悉代码的操作，如果不是的话，现在正是回去复习关键知识点的好时候。

  ```
  $('input.numeric').live('keypress',function(event){
    if (event.which < 48 || event.which > 57) return false;
  });
  ```

- 如前所述，没有验证日期字段。如何才能够确保只输入有效的日期（所选择的任何格式）呢？我们不可能像对待数字字段那样逐个字符来判断。

  在本书的后面，我们将看到 jQuery UI 插件如何巧妙地解决了这个问题，但是现在先使用已有的事件处理知识来测试一下你是否已经掌握了它们。

- 当限定的字段被添加到过滤器时，用户必须单击其中的一个字段以便获取焦点。这不是太友好！为示例添加代码以便在添加新控件时将它们设置为焦点。

- 使用全局变量保存过滤器的个数违背了编码常规，并且还限制了一个页面只能有一个“部件”实例。通过为合适的元素应用 data()方法来替换全局变量，牢记我们可能要在页面上多次使用这个变量。

- ❑ 表单允许用户指定多个过滤器，但是没有定义如何将这些过滤器应用到服务器上。它们是应该形成一个逻辑或（必须匹配其中的一个过滤器），还是一个逻辑与（必须匹配其中所有的过滤器）呢？通常情况下，如果未指定则假设为逻辑或。如何改变表单以便允许用户指定采用哪个逻辑呢？
- ❑ 对于代码的健壮性和界面的易用性，你会做其他哪些改进呢？jQuery 如何帮忙呢？

如果你想出了比较得意的新点子，一定要访问 Manning 为本书准备的网站页面 http://www.manning.com/bibeault2，其中包含到论坛的链接。我们鼓励用户发表自己的解决方案，这样每个人都能看到并参与讨论！

4.4 小结

借助目前为止所学的 jQuery 知识，本章将我们引入了事件处理的世界。

我们了解到在 Web 页面中实现事件处理有一些棘手的挑战，但是这样的处理对于创建交互 Web 应用程序中的页面是必不可少的。这些挑战中一个重要的事实是，在现代流行的浏览器中有 3 个模型按照不同的方式运行。

传统的基本事件模型，也被非正式地称为 DOM 第 0 级事件模型，在声明事件监听器时享有某种程度上独立于浏览器的操作，但是监听器函数的实现需要依赖于浏览器的不同代码来处理 `Event` 实例中的差异。这个事件模型可能是页面开发者最熟悉的，它通过将监听器函数的引用赋值给元素的属性（例如 `onclick` 属性）来创建 DOM 元素的事件监听器。

这个模型虽然简单，但是却限制了在特定的 DOM 元素上的任何一个事件类型都只能有一个监听器。

使用更加先进和标准化的 DOM 第 2 级事件模型可以来避免这方面的不足，在这个模型中我们使用一个 API 将处理器绑定到其事件类型和 DOM 元素上。尽管这个模型很通用，但是它仅支持标准兼容的浏览器，例如 Firefox、Safari、Camino 以及 Opera。

对于 IE 浏览器，即使直到 IE8 还是使用基于 API 的专有事件模型，它提供了 DOM 第 2 级事件模型的功能的子集。

使用一系列的 `if` 语句编写所有的事件处理代码（一个子句用于标准的浏览器，另一个用于 IE 浏览器），这会使我们患上早期痴呆症。幸运的是，jQuery 将我们从这种命运中拯救了出来。

jQuery 提供了通用的 `bind()`方法用来在元素上建立任意类型的事件监听器，以及特定事件的便捷方法，例如 `change()`和 `click()`。这些方法以独立于浏览器的方式运行，并且使用在事件监听器中最常用到的标准属性和方法来标准化传入的 `Event` 实例。

jQuery 也提供了删除事件处理器或者在脚本控制下触发这些处理器的方法，并且还定义了一些更高级的方法来尽可能简单地实现普通的事件处理任务。

这一切似乎还不够，jQuery 还提供了 `live()`方法，为可能还尚不存在的元素预先指定处理器，而且允许我们指定自定义方法来轻松地注册将会在自定义事件发布时调用的处理器。

我们探讨了在页面上使用事件的一些示例，还研究了利用了很多已学过的概念创建的综合示例。在下一章，我们将会分析 jQuery 如何以这些能力为基础来引入动画和动画特效。

第 5 章

用动画和特效装扮页面

本章内容

- ❑ 不使用动画显示和隐藏元素
- ❑ 使用核心动画特效显示和隐藏元素
- ❑ 编写自定义动画
- ❑ 控制动画和函数队列

自 LiveScript（LiveScript 随后被命名为 JavaScript）出现以来，浏览器走过了一段漫长的道路，并于 1995 年被引入 Netscape Navigator，以便为 Web 页面添加脚本控制。

在早期的日子里，页面开发者拥有的能力非常有限，不仅是因为 API 很少，而且也是因为脚本引擎低效和系统的性能不高。使用有限的功能来实现动画和特效的想法是可笑的，多年来唯一的动画是由带动画效果的 GIF 图片实现的（非常不好用，还会降低页面可用性，令人反感）。

哦，时代已经改变了。如今的浏览器脚本引擎快如闪电，运行在 10 年前不可想象的硬件上，为页面开发者提供了各种丰富的功能。

但尽管如此，由于动画的实现依赖于低级别的操作，而 JavaScript 没有易用的动画引擎，因此我们只能依靠自己。别紧张，jQuery 为我们提供了相当简单的接口来创建各种优雅的特效。

但是在研究如何为页面添加很炫的特效之前，需要思考一个问题，应该添加吗？就像一部充斥着特效而没有情节的好莱坞大片一样，一个滥用特效的页面将起到意想不到的负面作用。注意，特效应该是用来增强页面的可用性，而不是用来炫耀的。

记住这一点，下面来看看 jQuery 提供了哪些功能。

5.1 显示和隐藏元素

我们将要对一个或一组元素上应用的最普通的动态特效类型，可能就是显示或隐藏元素的简单操作。我们随后将看到更多花哨的动画（例如使元素淡入或淡出），但是有时我们想尽量保持简单，也就是让元素立即呈现或者立即消失！

显示和隐藏元素的方法和我们所料的一样：`show()`用于显示包装集中的元素，而`hide()`用于隐藏它们。我们打算推迟介绍这两个方法的正式语法，稍后你就会明白其中的缘由。现在集中注意力来看下如何不带参数地使用这些方法。

虽然这些方法可能看起来简单，但还是应该牢记几件事情。首先，jQuery 通过将元素的 style.display 属性改为 none 来隐藏元素。如果包装集中的一个元素已经处于隐藏状态，它将继续保持隐藏状态，但仍然会被返回给 jQuery 链。例如，假设有如下 HTML 片段：

```
<div style="display:none;">This will start hidden</div>
<div>This will start shown</div>
```

如果应用`$("div").hide().addClass("fun")`，将得到如下结果：

```
<div style="display:none;" class="fun">This will start hidden</div>
<div style="display:none;" class="fun">This will start shown</div>
```

注意，虽然第一个元素已经隐藏了，但它仍然是匹配集的一部分，并且参与方法链的剩余操作。

其次，jQuery 可以将 display 属性从 none 改变为 block 或 inline 来显示对象。选择哪个值取决于之前是否显式地设置了元素的 display 属性值。如果显式地设置了这个值，则它将被记录下来并被恢复。否则就取决于目标元素类型的 display 属性的默认状态。例如，`<div>`元素的 display 属性的默认值为 block，而`<span>`元素的 display 属性的默认值为 inline。

下面看看这些方法的用法。

5.1.1　实现可折叠的“模块”

毫无疑问，你很熟悉那些从其他站点聚合数据的站点，这些站点在某种“主面板”页面将各种信息呈现在可配置的“模块”中。iGoogle 站点就是一个很好的例子，如图 5-1 所示。



图 5-1　iGoogle 就是这类站点的一个例子，它将聚合的信息在一系列的主面板模块中显示出来

这个站点允许我们对页面如何呈现进行很多配置，包括来回移动模块、将模块扩大到整个页面尺寸、指定配置信息，甚至完全删除模块。但是有一件事情该站点不允许我们做（至少在编写本书时还不允许），就是“卷起”一个模块到其标题栏，这样可以占用更少的空间而无需从页面上删除这个模块[①]。

下面来定义自己的主面板模块，比 Google 先进的是允许用户卷起一个模块到其标题栏。

首先，来看下我们期望看到的模块在正常状态和卷起状态下的样子，分别如在图 5-2a 和图 5-2b 所示。

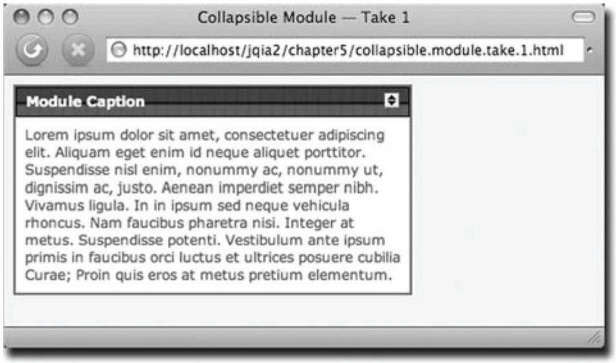


图 5-2a　创建自己的主面板模块，它包含两部分：一个包含标题和卷起按钮的工具栏以及一个可以展示数据的主体

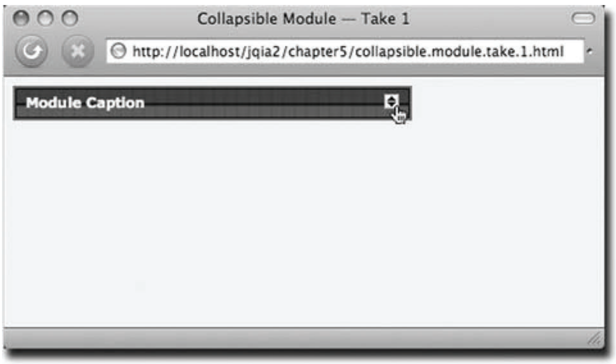


图 5-2b　当单击卷起按钮时，模块的主体消失了，就好像被卷入了标题栏一样

在图 5-2a 中，我们创建的模块包括两个主要部分：标题栏和主体。主体包含了模块的数据——在这种情况下，是随机的“Lorem ipsum”文本。标题栏更加有趣，它包含了模块的标题和一个小按钮，单击这个小按钮就会调用卷起（展开）功能。一旦单击了这个按钮，模块的主体将会消失，就好像被卷入标题栏一样。随后的单击会展开主体，恢复其初始外观。

① 这个说法是不准确的，折叠一个模块的功能就隐藏在标题栏向下箭头按钮的下拉菜单中。

用于创建模块结构的 HTML 标记是非常简单的。我们为元素应用了大量的类名用来标识元素以及调整 CSS 样式。

```
<div class="module">
  <div class="caption">
    <span>Module Caption</span>
    <img src="rollup.gif" alt="rollup" title="rolls up this module"/>
  </div>
  <div class="body">
    Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
    Aliquam eget enim id neque aliquet porttitor. Suspendisse
    nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
    Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
    sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
    Integer at metus. Suspendisse potenti. Vestibulum ante
    ipsum primis in faucibus orci luctus et ultrices posuere
    cubilia Curae; Proin quis eros at metus pretium elementum.
  </div>
</div>
```

整个结构包含在一个类名为 `module` 的`<div>`元素中，而标题和主体结构分别由类名为 `caption` 和 `body` 的`<div>`子节点来呈现。

为了给这个模块添加卷起行为，我们给标题栏中的图片配备一个负责所有特效的点击处理器。有了 `hide()`和 `show()`方法在手，为模块添加卷起行为比从耳朵后面拿出 25 美分[①]还要简单。

下面就是在就绪处理器中实现卷起行为的代码：

```
$('div.caption img').click(function(){                    ❶ 为按钮添加行为
  var body$ = $(this).closest('div.module').find('div.body');   ❷ 找出相关的主体
  if (body$.is(':hidden')) {
    body$.show();                                          ❸ 显示主体
  }
  else {
    body$.hide();                                          ❹ 隐藏主体
  }
});
```

可以想象，这段代码首先为标题栏上的图片创建单击处理器❶。

在单击处理器中，首先找到模块相关的主体。我们需要找到特定的模块主体，因为在主面板页面可能有很多模块，所以不能选择类名为 `body` 的所有元素。通过查找最近的 `module` 容器可以快速地找出正确的主体元素，然后把它作为 jQuery 上下文，使用下面的 jQuery 表达式来查找内部的 `body` 元素❷：

```
$(this).closest('div.module').find('div.body')
```

如果你对这个表达式如何查找正确的元素不是很清楚的话，现在正是时候来复习第 2 章中有关查找和选择元素的内容。

一旦找到了主体，决定主体是应该隐藏还是显示就非常简单了（jQuery 的 `is()`方法在这里显得非常方便），视情况使用 `show()`方法❸来显示元素或使用 `hide()`❹方法来隐藏元素。

---

① 这是一个常见的哄小孩儿玩的魔术，参考 http://www.wikihow.com/Pull-a-Coin-Out-Of-an-Ear。

注意　在这个代码示例中，当使用变量来存储包装集引用的时候，我们引入了一个很多人都采用的约定：在变量名中使用$字符。有的人将$作为前缀，有的人将$作为后缀（和我们在这里做的一样——如果你认为$代表了“包装器”，那么变量名 body$可以读作“主体包装器”）。在上述两种情况下，这都是一个很方便的方法，以便记住变量包含一个包装集的引用，而不是一个元素或者其他类型的对象。

这个页面的完整代码可以在文件 chapter5/collapsible.module.take.1.html 中找到，如代码清单 5-1 所示。（如果认为文件名中的“take 1”意味着我们会再次使用这个示例，那你是对的！）

代码清单 5-1　可折叠模块的首次实现

```
<!DOCTYPE html>
<html>
  <head>
    <title>Collapsible Module &mdash; Take 1</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css" />
    <link rel="stylesheet" type="text/css" href="module.css" />
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript">
      $(function() {

        $('div.caption img').click(function(){
          var body$ = $(this).closest('div.module').find('div.body');
          if (body$.is(':hidden')) {
            body$.show();
          }
          else {
            body$.hide();
          }
        });

      });
    </script>
  </head>

  <body class="plain">

    <div class="module">
      <div class="caption">
        <span>Module Caption</span>
        <img src="rollup.gif" alt="rollup" title="rolls up this module"/>
      </div>
      <div class="body">
        Lorem ipsum dolor sit amet, consectetuer adipiscing elit.
        Aliquam eget enim id neque aliquet porttitor. Suspendisse
        nisl enim, nonummy ac, nonummy ut, dignissim ac, justo.
        Aenean imperdiet semper nibh. Vivamus ligula. In in ipsum
        sed neque vehicula rhoncus. Nam faucibus pharetra nisi.
        Integer at metus. Suspendisse potenti. Vestibulum ante
        ipsum primis in faucibus orci luctus et ultrices posuere
        cubilia Curae; Proin quis eros at metus pretium elementum.
```

```
      </div>
    </div>

  </body>

</html>
```

这一点也不难，不是吗？但事实证明，它还可以更加简单！

5.1.2 切换元素的显示状态

在显示和隐藏之间切换元素的状态（正如之前为可折叠模块示例所做的那样）是一件司空见惯的事情，因此 jQuery 定义了 `toggle()`方法使其更加简单。

为可折叠模块应用 `toggle()`方法，观察它是如何简化代码清单 5-1 中的代码。代码清单 5-2 只显示了重构页面的就绪处理器（不需要其他的改变），改动的部分以粗体突出显示。完整的页面代码可以在文件 chapter5/collapsible.module.take.2.html 中找到。

代码清单 5-2 使用 `toggle()`简化的可折叠模块的代码

```
$(function() {

  $('div.caption img').click(function(){
    $(this).closest('div.module').find('div.body').toggle();
  });

});
```

注意，我们不再需要使用条件判断语句来决定是应该隐藏还是显示模块的主体；`toggle()` 为我们完成了切换显示状态的任务。这允许我们在很大程度上简化代码，同时也不需要将主体引用保存在变量中了。

使用这个方法让元素瞬间显示和消失非常方便，不过有时我们希望这种转变不要太突然。下面来看看有哪些可用的功能。

5.2 用动画改变元素的显示状态

人类的认知能力就是这样，元素项的突然出现或者消失会让人不安。如果在错误的时间眨了下眼睛，就可能会错过转变的过程，留下来的疑问是：“刚才发生了什么？”

短时间的渐变过程有助于我们了解什么正在改变，以及如何从一个状态转换到另一个状态——这也正是 jQuery 核心特效起作用的地方。总共有 3 组特效类型：

- 显示和隐藏（在 5.1 节中显示和隐藏方法还有一些没有介绍的内容）；
- 淡入和淡出；
- 滑下和滑上。

下面逐一考察每一组特效。

5.2.1 渐变地显示和隐藏元素

`show()`、`hide()`和 `toggle()`方法实际上比上一节介绍的要复杂一点。当不带参数调用时，

5

这些方法实现对包装元素显示状态的简单操作，使这些元素从显示器上瞬间显示或者隐藏。但是当传入参数时，这些效果是用动画方式来呈现的，因此受影响元素的显示状态将在短时间内持续改变。

接着来看下这些方法的完整语法。

方法语法：hide

hide(speed, callback)

使包装集中的元素变成隐藏状态。如果不带参数调用，则通过设置元素的 display 样式属性值为 none 使操作在瞬间完成。如果指定了 speed 参数，则通过调整元素的宽度、高度和不透明度为零，使元素在一段时间内逐渐隐藏，在元素完全隐藏后再设置它们的 display 样式属性为 none，从显示器上删除元素。

可选的函数，指定在动画完成时调用的回调函数

参数

speed	（数字|字符串）可选项，用于指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则不会产生动画，并且立即从显示器上删除元素
callback	（函数）可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

方法语法：show

show(speed, callback)

使包装集中任何隐藏的元素显示出来。如果不带参数调用，则设置元素的 display 样式属性值为一个合适的值（block 或者 inline）使操作在瞬间完成。如果指定了 speed 参数，则通过调整使元素的宽度、高度变为全尺寸并提高不透明度，使元素在一段指定的时间内逐渐显示出来。

可选的函数，指定当动画完成时调用的回调函数

参数

speed	（数字|字符串）可选项，用于指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则不会产生动画，并且元素会立即从显示器上显示出来
Callback	（函数）可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

方法语法：toggle

toggle(speed, callback)

在任何隐藏的包装元素上执行 show()，同时在任何非隐藏的包装元素上执行 hide()。参阅这两个方法的语法描述以便了解它们的语义

参数

speed	（数字|字符串）可选项，用于指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则不会产生动画
callback	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

我们可以使用 toggle()方法的另一个变体来进行更多的控制。

方法语法：toggle

toggle(condition)

基于传入条件的运算结果显示和隐藏匹配的元素。如果结果为 true，则显示元素；否则隐藏元素

参数

condition	（布尔）确定是显示元素（如果为结果 true），还是隐藏元素（如果结果为 false）

返回值

包装集

下面对可折叠模块进行第 3 次修改，以动画方式来展开和折叠各节。

有了前面的信息，你可能认为代码清单 5-2 需要进行的唯一修改是将 toggle()方法的调用方式改成：

```
toggle('slow')
```

你是对的。

但是没这么快！因为那太简单了，让我们借此机会为模块添加一些很炫的效果。

比方说，为了给用户一个明确无误的视觉线索，当模块处于折叠状态时，我们希望模块标题栏显示不同的背景。我们可以在动画开始前就做出改变，但是如果能在动画结束后实现改变会更加优雅。

我们不能在调用动画方法后立即进行处理，因为动画不会阻塞 JavaScript 的执行。与动画方法调用紧跟着的语句会立即执行，甚至有可能在动画开始之前执行。

不过，这就是使用 `toggle()`方法来注册回调函数的用武之地。

将要采取的方法是，在动画结束后为模块添加一个类名来识别其处于折叠状态，而在模块处于非折叠状态时删除此类名。CSS 规则将负责余下的细节。

你最初的想法可能是使用 `css()`为标题栏直接添加一个 `background` 样式属性，但这简直就是杀鸡用牛刀。

模块标题“正常状态”的 CSS 规则（可以在文件 chapter5/module.css 中找到）如下所示：

```
div.module div.caption {
  background: black url('module.caption.backg.png');
  ...
}
```

我们也添加了另外一个规则：

```
div.module.rolledup div.caption {
  background: black url('module.caption.backg.rolledup.png');
}
```

只要父模块拥有 `rolledup` 类，第二个规则就会改变标题栏的背景图片。因此，为了改变显示效果，所要做的就是在适当的时间点添加或删除模块的 `rolledup` 类。

代码清单 5-3 显示了修改后的就绪处理器的代码。

代码清单 5-3 模块的动画版本，拥有可以如魔术般进行改变的标题栏

```
$(function() {

  $('div.caption img').click(function(){
    $(this).closest('div.module').find('div.body')
      .toggle('slow',function(){
        $(this).closest('div.module')
          .toggleClass('rolledup',$(this).is(':hidden'));
       });
  });

});
```

修改后的页面可以在文件 chapter5/collapsible.list.take.3.html 找到。

我们知道有很多像我们一样爱修修改改的人，为此创建了一个方便的工具，可用来进一步研究这些工具的用法和剩余的特效方法。

引入 jQuery 特效实验室页面

在第 2 章中，我们引入了实验室页面的概念，以便对 jQuery 选择器进行练习。本章，我们创建了一个 jQuery 特效实验室页面 chapter5/lab.effects.html，可以用来探索 jQuery 特效的操作。

在浏览器中加载此页面，结果如图 5-3 所示。

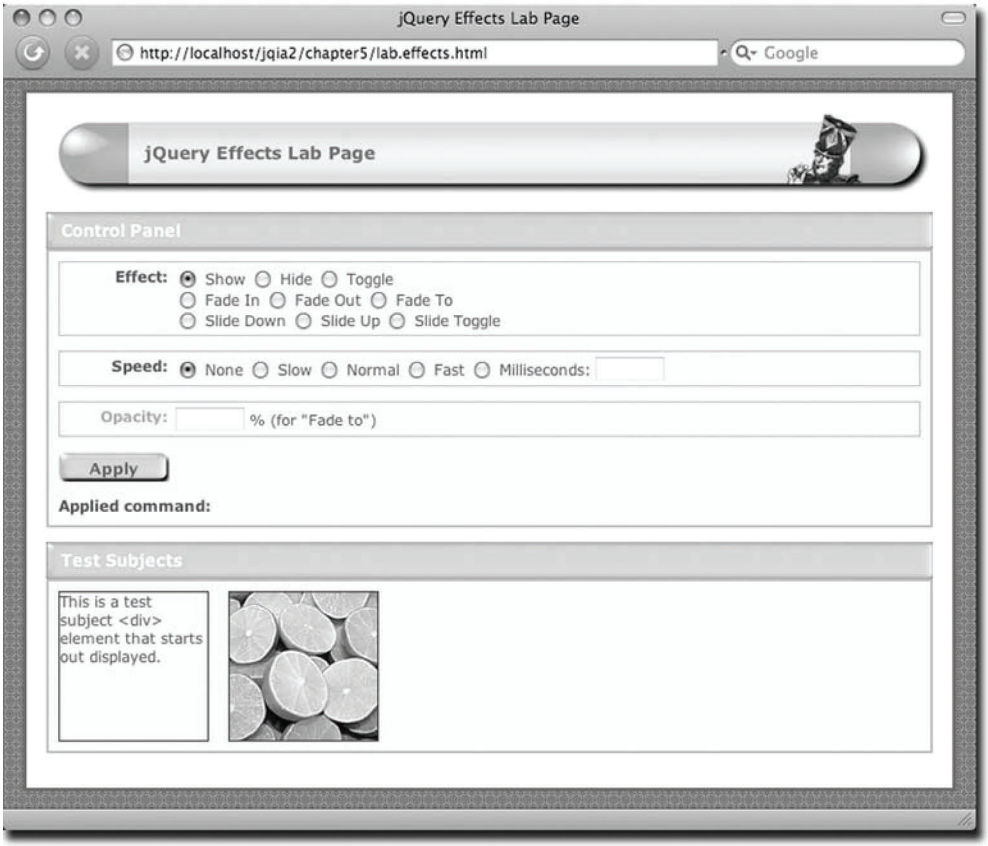


图 5-3 jQuery 特效实验室页面的初始状态，用来协助研究 jQuery 特效方法的操作

这个实验室页面包含两个主要面板：控制面板用来指定将要应用的特效，另一个面板包含将要应用特效的 4 个测试对象元素。

“他们不会这么笨吧？”你可能会想。“明明只有两个测试对象。”

不，本书的作者没有遗漏任何内容。的确有 4 个元素，但是其中的 2 个元素（一个是用来显示文本的<div>元素，另一个是图片元素）在初始时是隐藏的。

下面使用这个页面来演示下目前为止已讨论方法的运行过程。在浏览器中加载此页面，然后跟着做下面的练习。

- **练习 1**——初始页面加载后保持控件的状态不变，然后单击 Apply 按钮。这将会不带参数地执行 `show()`方法。应用的语句显示在 Apply 按钮下面供参考。注意那 2 个初始隐藏的测试对象元素是如何立即显示的。如果你很好奇为什么最右侧的皮带图片看起来有点褪色，那是因为我们有意地设置它的不透明度为 50%。
- **练习 2**——选择 Hide 单选按钮，然后单击 Apply 来执行无参数的 `hide()`方法。所有的测

试对象立即消失了。请特别注意，这些对象原来所在的面板收了起来。这表明这些元素已经从显示器上完全删除，而不是被隐藏起来。

注意　当说到某个元素被从显示器中删除时（在这里，以及其余关于特效的讨论中），意思是该元素不再被浏览器的布局管理器考虑在内，这是通过设置该元素的 CSS 的 `display` 样式属性为 `none` 来实现的。这并不意味着元素从 DOM 树中删除，没有哪个特效可以将元素从 DOM 中删除。

- **练习 3**——选择 Toggle 单选按钮，然后单击 Apply。再次单击 Apply，继续单击。你将会注意到每次执行 `toggle()`都将切换测试对象的显示状态（显示或隐藏）。
- **练习 4**——重新加载页面以便使所有控件恢复到它们的初始状态（在 Firefox 或者其他 Gecko 浏览器中，请将焦点移到地址栏上，然后单击 Enter 键——只单击重新加载按钮不会重置表单元素[①]）。选择 Toggle，然后单击 Apply。注意初始时可见的两个对象是如何消失的以及隐藏的两个对象是如何显示出来的。上述过程证明了 `toggle()`方法单独作用于每一个包装元素，显示那些隐藏的元素并隐藏那些显示的元素。
- **练习 5**——我们将在这个练习中进入动画领域。刷新页面，保持 Show 的选中状态，并设置 Speed 为 Slow。单击 Apply，并仔细观察测试对象。那两个隐藏的元素不是突然出现，而是从左上角逐渐伸展开来。如果你想真正看到发生了什么事情，那请刷新页面，选择 Milliseconds（毫秒）来设置速度并输入 10 000 作为速度值。这将把特效的持续时间延长到（令人心烦的）10 秒，这样就有足够的时间来观察特效的行为。
- **练习 6**——选择 Show、Hide 和 Toggle 的不同组合以及不同的速度来测试这些特效，直到你感觉已经很好地掌握了对它们的操作。

有了 jQuery 特效实验室页面以及第一组特效是如何运行的知识，下面来看看下一组特效。

5.2.2　使元素淡入和淡出

如果仔细观察过 `show()`和 `hide()`特效的运行过程，你会注意到它们不仅缩放元素的尺寸（扩大或收缩到合理尺寸），而且在元素扩大或者缩小时会调整不透明度。下一组特效 `fadeIn()`和 `fadeOut()`只会影响元素的不透明度。

除了缺少缩放功能，这些方法的工作方式类似于 `show()`和 `hide()`的动画形式。它们的语法如下所示。

---

① 这是由于 Firefox 中表单自动完成导致的问题，可以设置表单的 `autocomplete` 特性为 `off` 来修正这个问题，参考：https://developer.mozilla.org/en/How_to_Turn_Off_Form_Autocompletion。

方法语法：fadeIn

`fadeIn(speed, callback)`

通过将元素的不透明度提高到其初始值来使所有匹配的隐藏元素显示出来。这个值要么是最初应用到元素的不透明度，要么是 100%。不透明度改变的持续时间由 `speed` 参数决定。只有隐藏的元素受影响

参数

`speed`	（数字|字符串）指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：`slow`、`normal` 或者 `fast`。如果省略此参数，则默认为 `normal`
`callback`	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（`this`）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

方法语法：fadeOut

`fadeOut(speed, callback)`

通过将元素的不透明度降低到 0%，然后从显示器上删除该元素，来从显示器上删除任何显示的匹配元素。不透明度改变的持续时间由 `speed` 参数决定。只有显示的元素受影响

参数

`speed`	（数字|字符串）指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：`slow`、`normal` 或者 `fast`。如果省略此参数，则默认为 `normal`
`callback`	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（`this`）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

让我们使用 jQuery 特效实验室页面实现更多有趣的效果。打开实验室页面，运行一组类似于前一节的练习题，不过这次选择 Fade In 和 Fade Out（现在还不用担心 Fade To，马上会介绍它）。

值得重点关注的是，当调整元素的不透明度时，jQuery 的 `hide()`、`show()`、`fadeIn()`以及 `fadeOut()`特效会记住元素不透明度的初始值并遵守这个值。在实验室页面中，在隐藏最右边的皮带图片之前，我们有意地设置它的初始不透明度为 50%。在应用 jQuery 特效改变不透明度的整个过程中，它的初始值从来就没有被覆盖过。

在实验室页面多做些练习题，尽量熟练掌握消退特效的操作。

jQuery 通过 `fadeTo()`方法提供了另一个特效。这个特效调整元素的不透明度，和前面考察过的消退特效一样，但是它绝对不会从显示器上删除元素。在使用实验室页面练习 `fadeTo()`方法之前，先来看下它的语法。

方法语法：fadeTo

fadeTo(speed, opacity, callback)

逐渐改变包装元素的不透明度，从它们的当前值到 opacity 指定的新值

参数

speed	（数字|字符串）指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则默认值为 normal
opacity	（数字）将要调整的元素的目标不透明度，指定的取值范围是 0.0~1.0
callback	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

与其他特效在隐藏和显示元素的同时调整不透明度不同，`fadeTo()`不会记住元素的初始不透明度。这是合理的，因为这个特效的根本目的就是为了显式地改变不透明度到指定的值。

在浏览器下打开实验室页面，让所有元素都显示出来（现在你应该知道怎么做了）。然后做下面的练习。

- **练习 1**——选择 Fade To 和一个足够慢的速度值以便观察行为，可以设置为 4000 毫秒。现在设置 Opacity 字段（期望值为 0~100 之间的整数，当传入方法时会转换为 0.0~1.0 之间的值）为 10，然后单击 Apply。测试对象将会在 4 秒的时间内淡出至 10%的不透明度。
- **练习 2**——设置不透明度为 100，然后单击 Apply。所有的元素（包括初始半透明的皮带图片）都被调整为完全不透明。
- **练习 3**——设置不透明度为 0，然后单击 Apply。所有的元素都淡出为不可见状态，但是注意一旦它们消失后，包含这些元素的模块并没有向上收起。和 `fadeOut()`特效不同，`fadeTo()`绝对不会从显示器上删除元素，即使这些元素已经完全不可见了。

继续练习 Fade To 特效，掌握它的工作原理，然后准备学习下一组特效。

5.2.3　上下滑动元素

另外一组用于隐藏和显示元素的特效（slideDown()和 slideUp()）与 hide()和 show()的工作方式也很类似，只不过当元素在显示的时候好像是从顶部滑下，而在隐藏的时候好像是滑入顶部。

与 hide()和 show()一样，滑动特效也有一个相关的方法来切换元素的隐藏和显示状态：slideToggle()。你应该已经很熟悉这类方法的语法了，如下所示。

方法语法：slideDown

slideDown(speed, callback)

通过逐渐增加元素的垂直尺寸来使所有匹配的隐藏元素显示出来。只有隐藏的元素受影响

参数

speed	（数字|字符串）指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则默认为 normal
callback	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

方法语法：slideUp

slideUp(speed, callback)

通过逐渐减少元素的垂直尺寸来从显示器上删除匹配的显示元素

参数

speed	（数字|字符串）指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则默认为 normal
callback	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

5

方法语法：slideToggle

slideToggle(speed, callback)

在任何隐藏的包装元素上执行 slideDown()，同时在任何显示的包装元素上执行 slideUp()。请参阅这两个方法的语法描述以便了解它们的语义

参数

speed	（数字|字符串）可选项，用于指定特效持续的时间，可以是若干毫秒数，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略此参数，则默认为 normal
callback	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数

返回值

包装集

除了显示和隐藏元素的方式不同之外，这些特效和其他显示和隐藏特效非常相似。为了明确理解这一点，打开 jQuery 特效实验室页面，运行一些类似于之前应用了其他特效的练习题。

5.2.4　停止动画

有时我们可能出于某种原因在动画开始后就终止动画。这可能是由于一个用户事件表明有别的事情将要发生，或者是因为想要开始一个全新的动画。stop()命令可以实现这个目的。

方法语法：stop

stop(clearQueue, gotoEnd)

停止匹配集元素上正在进行的所有动画

参数

clearQueue	（布尔）如果指定为 true，则不仅停止当前动画，而且停止在动画队列中正在等待执行的所有其他的动画（动画队列？很快就会介绍到……）
gotoEnd	（布尔）如果指定为 true，则使当前动画执行到其逻辑结束（而不是仅仅停止动画）

返回值

包装集

注意，对动画元素所做的任何改变都将继续有效。如果希望恢复元素的初始状态，那我们就需要通过 css()方法或者类似的方法将元素的 CSS 值重置为其初始值。

顺便提一下，可以使用一个全局标志来完全禁止所有的动画。设置 jQuery.fx.off 标志为 true 将导致所有的特效立即发生，而没有动画过程。我们将在第 6 章中正式介绍这个标志和其他的 jQuery 标志。

目前为止我们已经了解了核心 jQuery 的内置特效，下面来研究如何编写自定义的特效！

5.3 创建自定义动画

jQuery 有意地保持少量核心特效的目的是为了尽量减少核心 jQuery 的体积，同时期望页面开发者使用插件（包括将在第 9 章中开始探讨的 jQuery UI）来酌情添加更多的动画。编写自定义动画非常简单。

jQuery 公开了 `animate()`包装器方法，它允许为包装集中的元素应用自定义动画特效。来看下它的语法。

方法语法：animate

animate(properties, duration, easing, callback)

animate(properties, options)

将 properties 和 easing 参数指定的动画应用到包装集中的所有成员上。可以指定一个回调函数（可选），在动画完成时调用。另一种格式除了指定 properties 之外，还指定了一组 options

参数

properties	（对象）一个散列对象，指定动画结束时所支持的 CSS 样式应该达到的值。通过调整元素样式属性的当前值到此散列对象中指定的值来产生动画（确保在指定多字属性时使用骆驼拼写法[①]。）
duration	（数字|字符串）可选项，用于指定特效持续的时间，可以是若干毫秒，也可以是以下预定义字符串之一：slow、normal 或者 fast。如果省略或者指定为 0，则不会有动画过程，并且立即将元素指定的 properties 同步地设置为目标值
easing	（字符串）一个函数的名称（可选），用于指定动画缓动特效。缓动函数必须通过名称来注册，通常由插件提供。核心 jQuery 提供了两个缓动函数，分别注册为 linear 和 swing（可查看第 9 章里由 jQuery UI 提供的缓动函数列表。）
callback	（函数）一个可选的函数，当动画结束时调用。不向此函数传递任何参数，但是将函数上下文（this）设置为执行动画的元素。为经历动画的每个元素调用此回调函数
options	（对象）使用一个散列对象来指定动画参数值，支持的属性如下所示： ❑ duration——参考前面对 duration 参数的描述 ❑ easing——参考前面对 easing 参数的描述 ❑ complete——动画完成时调用的函数 ❑ queue——如果为 false，则动画立即运行，无需排队 ❑ step——在动画的每一步都会调用的回调函数。步骤序号和内部特效对象（其中不会包含太多页面开发者感兴趣的东西）将被传入此回调函数。函数的上下文被设置为正在执行动画的元素

返回值

包装集

① 这里指的是一个词首字母小写，后面每个词首字母大写的写法，比如 borderWidth, marginLeft。

我们可以通过提供一套 CSS 样式属性以及目标值（随着动画的进行，这些属性将向此目标值聚集）来创建自定义属性。动画从元素的初始样式值开始，向目标值的方向调整样式值来运行。特效过程中样式的中间值（由动画引擎自动处理）是由动画的持续时间和缓动函数决定的。

指定的目标值可以是绝对值也可以是相对于起点的值。为了指定相对值，要为目标值添加`+=`或者`-=`前缀，来分别表明正方向或负方向的相对目标值。

术语缓动（easing）用来描述动画的过程和帧速度的处理方式。通过对动画持续时间和当前时间点应用一些花哨的数学公式，可以让特效产生一些有趣的变化。编写缓动函数是一个复杂的小众主题，通常只有最铁杆的插件开发者才会对此感兴趣。本书中，我们不打算深入探讨自定义缓动函数这个主题。在第 9 章中研究 jQuery UI 时，我们将看到更多的缓动函数。

在默认情况下，动画会被添加到执行队列中（随后会对此进行详细介绍）。为同一个对象应用多个动画会促使它们依次运行。如果希望以并行的方式运行动画，可以将`queue`选项设置为`false`。

可以用动画方式展现的 CSS 样式属性列表局限于能够接受数字值的属性，因为对这些属性而言，要有一个从初始值到目标值的逻辑递进过程。限定为数字值是完全合理的——我们如何设想一个非数值的属性（例如`background-image`）从开始值到结束值的逻辑过程呢？对于表示尺寸的值，jQuery 假设其默认的单位是像素，但是也可以通过包含`em`或者`%`后缀来指定`em`单位或者百分比。

常见的动画样式属性包括`top`、`left`、`width`、`height`和`opacity`。不过只要我们想实现的特效有意义，数字值的样式属性例如字体大小、外边距、内边距以及边框尺寸也能够以动画方式运行。

注意　jQuery UI 为指定颜色值的 CSS 属性添加了动画能力。在第 9 章讨论 jQuery UI 特效时会了解这些细节。

除了指定目标属性的值，我们也可以指定如下字符串之一：`hide`、`show`或者`toggle`。jQuery 将根据这个字符串的规范来计算出合适的结束值。例如，指定`opacity`属性为`hide`，将会导致元素的不透明度减少为`0`。使用这些特殊字符串中的任意一个都会有额外的效果，将自动从显示器上显示或者删除元素（就像`hide()`和`show()`方法一样），需要注意的是`toggle`会记住初始状态，因此随后指定目标属性值为`toggle`会恢复元素的初始状态。

当我们介绍核心动画时，你注意到消退特效的切换方法了吗？使用`animate()`和`toggle`来创建一个简单的自定义动画可以很容易地解决这个问题，如下所示：

```
$('.animateMe').animate({opacity:'toggle'},'slow');
```

将这句代码引入下一个逻辑步骤（创建一个包装器函数），编写代码如下：

```
$.fn.fadeToggle = function(speed){
  return this.animate({opacity:'toggle'},speed);
};
```

接下来再编写几个自定义动画。

5.3.1　自定义缩放动画

考虑一个简单的缩放动画，用来将元素的尺寸调整为其原始尺寸的两倍。下面编写该动画的代码，如代码清单 5-4 所示。

代码清单 5-4　自定义缩放动画

```
$('.animateMe').each(function(){          ❶ 遍历每一个匹配的元素
  $(this).animate({
      width: $(this).width() * 2,
      height: $(this).height() * 2          ❷ 指定单个的目标值
    },
    2000                        ❸ 设置持续时间
  );
});
```

为了实现这个动画，使用 each()来遍历包装集中的所有元素，以便为每个匹配元素单独应用动画❶。这非常重要，因为需要为每个元素指定的属性值是基于该元素的特有尺寸的❷。如果我们总是知道对单个元素应用动画（正如使用 id 选择器一样）或者对每个元素应用完全相同的一组值，那就无需使用 each()而直接对包装集应用动画。

在迭代函数中，animate()方法被应用到（通过 this 指向的）元素上，同时 width 和 height 样式属性值设置为元素初始尺寸的两倍大小。结果在两秒钟的时间内（将 duration 参数值指定为 2000❸），多个包装元素（或者单个元素）将会从其原始尺寸增长到该尺寸的两倍大小。

现在，来尝试一些更加华丽的特效。

5.3.2　自定义掉落动画

假设我们要以引人注意的动画方式从显示器上删除元素，这可能是因为要传达给用户的信息非常重要：删除的项一去不返，因此应该对要删除的项确认无误。我们用动画表达这个意思，元素从页面上掉落下来，同时逐渐从显示器上消失。

如果仔细想一下就能够知道，可以通过调整元素的 top 位置使其向页面底部移动来模拟掉落过程，同时通过调整 opacity 使元素看起来好像消失了一样。最后，当所有的事情都完成了，就可以从显示器上删除元素（类似于 hide()的动画方法）。

我们可以使用代码清单 5-5 中的代码来实现这个掉落特效。

代码清单 5-5　自定义掉落动画

```
$('.animateMe').each(function(){
  $(this)                                   ❶ 使元素脱离静态流布局
    .css('position','relative')
    .animate(
      {
        opacity: 0,
```

```
        top: $(window).height() - $(this).height() -        ❷ 计算下降距离
             $(this).position().top
      },
      'slow',                                              ❸ 从显示器上删除元素
      function(){ $(this).hide(); }
    );
});
```

相比前一个自定义特效，这里多了一些处理步骤。再次遍历元素集合，调整元素的位置和不透明度。但是为了相对于初始位置来调整元素的 top 值，首先需要改变它的 CSS position 样式属性值为 relative❶。

然后，为动画过程指定目标 opacity 值为 0 和计算后的 top 值。我们不希望元素在页面上向下移动时超出窗体底部的范围，因为这会导致出现以前不曾出现过的滚动条，很可能会分散用户的注意力。我们不希望用户将注意力从动画上移开——吸引用户的注意力是当初采用动画的原因！因此我们使用元素的高度、垂直位置以及窗体的高度来计算出元素应该在页面中掉落的距离❷[①]。

当动画完成时，我们想从显示器上删除元素，因此指定了一个回调函数❸并在这个回调函数中为元素（以函数上下文的形式存在）应用 hide()方法的非动画版本。

注意　我们在这个动画中做了一些不必要的工作，目的是为了证明可以在回调函数中完成一些需要等到动画结束才能做的事情。如果指定 opacity 属性的值为 hide 而不是 0，那么在动画结束时将会自动删除元素，这样就无需使用回调函数了。

现在接着尝试更多“让它消失”的特效类型吧。

5.3.3　自定义消散动画

假设我们想要一种特效，让元素像一股烟一样消散在晴空中而不是从页面上掉落下来。为了通过动画来实现这个特效，我们可以组合使用比例缩放特效和不透明度特效，在使元素边放大边淡出。在这个特效中需要处理的一个问题是，如果固定元素的左上角并让元素在原地增大，那么这个消散过程就瞒不过用户的眼睛。我们想让元素的中心在增大过程中原地不动，因此作为动画的一部分除了调整元素的大小外还需要调整元素的位置。

消散特效的代码如代码清单 5-6 所示。

代码清单 5-6　自定义消散动画

```
$('.animateMe').each(function(){
  var position = $(this).position();
```

① 注意，这个计算过程只有在元素的 offsetParent()为 document.body 并且页面没有出现滚动条的情况下才正确。

```
  $(this)
    .css({position: 'absolute',
          top: position.top,
          left: position.left})          ❶ 使元素脱离静态流布局
    .animate(                            ❷ 调整元素的大小、位置和不透明度
      {
        opacity: 'hide',
        width: $(this).width() * 5,
        height: $(this).height() * 5,
        top: position.top - ($(this).height() * 5 / 2),
        left: position.left - ($(this).width() * 5 / 2)
      },
      'normal');
});
```

在这个动画中，我们将不透明度减少到 0 的同时把元素的尺寸增大为其原始尺寸的 5 倍，并根据新尺寸的一半大小来调整元素的位置，结果使得元素的中心保持原地不动❷[①]。在目标元素增大的过程中，我们不希望动画元素周围的元素被向外挤，因此通过修改目标元素的定位为 `absolute` 并且显式地设置其位置坐标❶，从而把动画元素从流布局中完全提取出来。

因为我们已经指定 `opacity` 值为 `hide`，因此一旦动画结束元素就会自动隐藏（从显示器中删除）。

可以通过加载页面 chapter5/custom.effects.html 来观察这 3 种自定义特效，如图 5-4 所示。

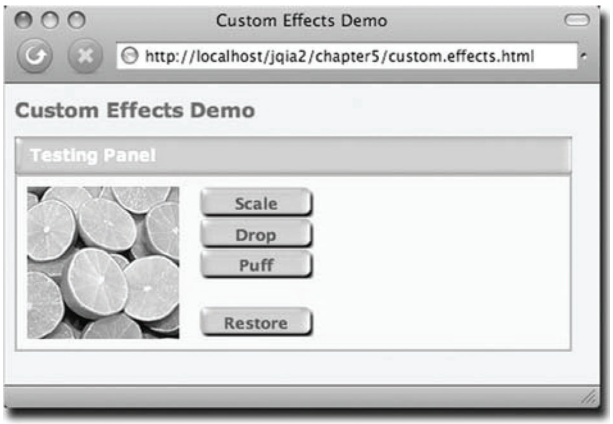


图 5-4 可以通过本示例页面提供的按钮来观察已开发的自定义特效 Scale（放大）、Drop（降落）和 Puff（消散）的运行

为了显示截图我们有意使浏览器窗口保持最小尺寸，但是为了正确地观察特效的行为，在运行本页面时你可能想让窗口更大一点。虽然我们很想向你展示这些特效的行为，但是截图有明显的局限性。不过图 5-5 显示了进行中的消散特效。

① 严格地说这个计算不正确，你可以仔细观察实验室页面中 Puff 的动画过程，会发现元素的中心并不是呆在原地。正确的代码应该是 `position.top - ($(this).height() * 4 / 2)`。

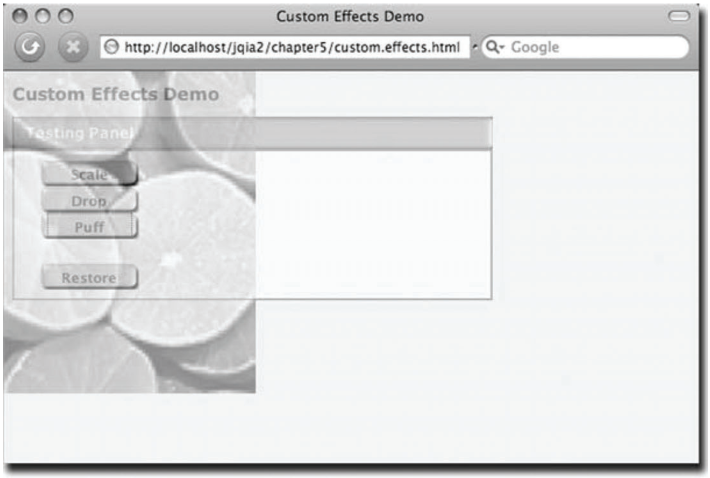


图 5-5　Puff 特效在减少不透明度的同时扩展和移动图片

接下来你可以尝试本页面中的各种不同的特效，并观察它们的行为。

目前为止已研究的所有示例都只使用了一个动画方法。下面来探讨当使用一个以上的动画方法时它们是如何工作的。

5.4　动画和队列

我们已经看到了如何使用单个动画方法为元素的多个属性应用动画，但是还没有真正研究过同时调用多个动画方法时这些动画的行为。

本节将探讨动画一起出现时的行为。

5.4.1　并发的动画

如果执行如下代码，你觉得会发生什么呢？

```
$('#testSubject').animate({left:'+=256'},'slow');
$('#testSubject').animate({top:'+=256'},'slow');
```

我们知道 animate()方法创建的动画在页面运行时不会阻塞代码的执行，其他的动画方法也是一样。可以通过如下的代码块来证明：

```
say(1);
$('#testSubject').animate({left:'+=256'},'slow');
say(2);
```

回想下在第 4 章介绍的 say()函数，它通过在页面的“控制台”上输出信息来避免弹出警告对话框（这肯定会破坏我们对实验示例的观察）。

如果执行这段代码，我们会发现“1”和“2”这两个消息没有等到动画完成就立即被一前一后地输出了。

那么，如果运行调用两个动画方法的代码时，会发生什么结果呢？由于第一个方法不会阻塞第二个，因此显而易见这两个动画会同时（或者之间相差几毫秒）发生，应用到测试对象上的特效将会是这两个特效的组合。在这个示例中，由于一个特效是调整 left 样式属性，另一个是调整 top 样式属性，因此预期结果是测试对象沿着一条蜿蜒的对角线运行。

下面来测试一下。在文件 chapter5/revolutions.html 中我们已经做了一个实验，其中创建了两个图片（其中只有一个是可以应用动画的），一个用来开始实验的按钮和一个用于显示 say()函数输出的“控制台”。图 5-6 显示了页面的初始状态。

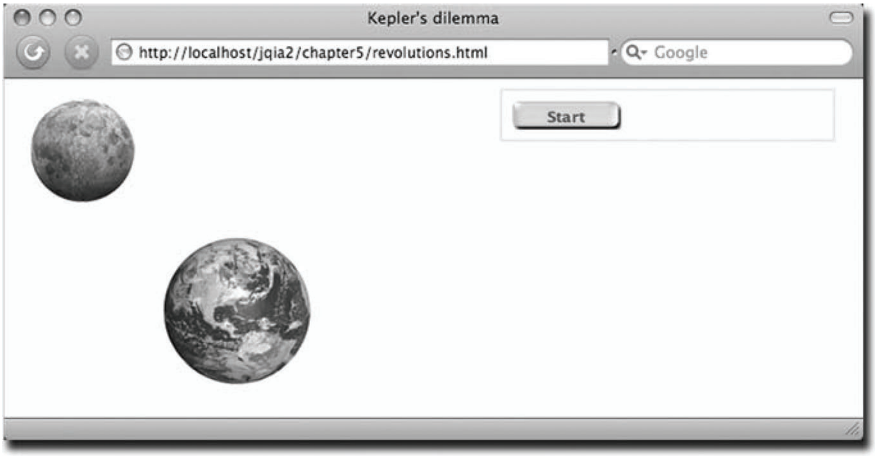


图 5-6　页面的初始状态，我们将在该页面上观察多个并发的动画

Start 按钮的实现逻辑如代码清单 5-7 所示。

代码清单 5-7　多个并发动画的实现逻辑

```
$('#startButton').click(function(){
  say(1);
  $("img[alt='moon']").animate({left:'+=256'},2500);
  say(2);
  $("img[alt='moon']").animate({top:'+=256'},2500);
  say(3);
  $("img[alt='moon']").animate({left:'-=256'},2500);
  say(4);
  $("img[alt='moon']").animate({top:'-=256'},2500);
  say(5);
});
```

在按钮的单击处理器中，我们一个接一个地调用了 4 个动画，其中还穿插了调用了 say()，用于指示什么时候触发动画的调用。

打开页面，然后单击 Start 按钮。

不出所料，控制台消息“1”到“5”会立即在控制台上显示出来，如图 5-7 所示，每个调用相对于前一个只有几毫秒的间隔。

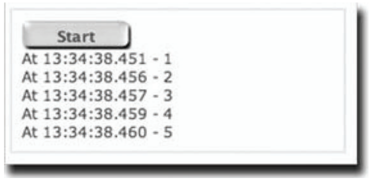


图 5-7　控制台消息接连不断地显示出来，说明在完成前没有任何动画方法被阻塞

但是动画效果如何呢？如果观察代码清单 5-7 中的代码就可以看到，有两个动画改变了 `top` 属性，另外两个动画改变了 `left` 属性。事实上，这些动画对每个属性的操作正好相反。那么结果如何呢？也许这些动画互相取消，使得 Moon（测试对象）保持原地不动？

不。一旦单击 Start，我们就会看到所有动画一个接一个地依次发生，因此 Moon 环绕 Earth 完成了一次完整的、有序的转动（尽管是在一个非常不自然的矩形轨道上运行，这应该会让开普勒（Kepler）感到崩溃）。

怎么回事？我们已经通过控制台消息来证明了动画不会阻塞，但是它们执行的顺序就像被阻塞了一样（至少两个动画相互之间是这样）。

真正发生的是，jQuery 在内部将动画放入队列并按顺序执行它们。

刷新 Kepler’s Dilemma 页面以便清空控制台，并连续单击 Start 按钮三次。（在两次单击之间停顿足够长的时间以防止双击。）你将会看到 15 条消息是如何被立即输出到控制台的，这表明单击处理器已经执行了三次，然后在等待 Moon 绕 Earth 运行 3 次的时侯你可以放松一下。

jQuery 将这 12 个动画进行排队并依次执行。jQuery 为此在每个动画元素上维持了一个名为 `fx` 的队列。（在下一节中将会详细介绍队列拥有名字的重要性。）

这种方式的动画队列意味着鱼和熊掌可以兼得。我们可以使用单个 `animate()`方法指定所有的动画属性来同时操作多个属性，也可以简单地按顺序调用动画来连续执行任何想要的动画。

更棒的是，jQuery 允许我们创建自己的执行队列，不仅仅为动画目的，还可以为任何其他目的。下面来继续学习这方面的内容。

5.4.2　将函数排队执行

函数队列的一个明显用途是将动画进行排队以便连续执行。但是这样做是否真正有利？毕竟，动画方法允许指定完成时回调，那为什么不在前一个动画的回调中执行下一个动画呢？

1. 添加函数到队列

回顾一下代码清单 5-7 中的代码（为了清晰起见去掉了 `say()`调用代码）：

```
$("img[alt='moon']").animate({left:'+=256'},2500);
$("img[alt='moon']").animate({top:'+=256'},2500);
$("img[alt='moon']").animate({left:'-=256'},2500);
$("img[alt='moon']").animate({top:'-=256'},2500);
```

将此代码与不使用函数队列，使用完成时回调的等效代码作对比：

```
$('#startButton').click(function(){
  $("img[alt='moon']").animate({left:'+=256'},2500,function(){
    $("img[alt='moon']").animate({top:'+=256'},2500,function(){
      $("img[alt='moon']").animate({left:'-=256'},2500,function(){
        $("img[alt='moon']").animate({top:'-=256'},2500);
      });
    });
  });
});
```

这段代码的回调变体不是太复杂，而且原始代码也很容易阅读（首先是编写更简单）。如果回调函数的主体非常复杂时……可以很容易看出，对动画进行排队使得代码的复杂度降低了。

如果想使自己的函数拥有相同的功能该怎么办呢？ jQuery 并不吝啬它的队列。我们可以将希望按顺序执行的任何函数进行排队来创建自己的队列。

可以在任何元素上创建队列，不同的队列通过各自唯一的名称来识别（除了 fx 被用于特效队列以外）。将函数实例添加到队列的方法是 queue()（并不奇怪），该方法有以下 3 种变体。

5

方法语法：queue

queue(name)

queue(name, function)

queue(name, queue)

第一种形式根据传入的名称查找建立在匹配集中第一个元素上的任意队列，并以函数数组的形式返回

第二种形式将传入的函数添加到匹配集中所有元素的命名队列的末尾。如果在某个元素上不存在这种命名队列，则创建一个队列

最后一种形式把匹配元素上现有的任意队列替换为传入的队列

参数

name	（字符串）需要获取的、向其中添加的或者替换的队列名称。如果省略，则默认为特效队列名称 fx
function	（函数）需要添加到队列结尾的函数。当调用此函数时，函数上下文（this）被设置为用于创建队列的 DOM 元素
queue	（数组）一个函数数组，用于替换命名队列中的现有函数

返回值

第一种形式返回函数数组，其余形式返回包装集

queue()方法常用于添加函数到命名队列的末尾，但是也可以用于获取队列中任意的现有函数或者替换队列中的函数列表。注意向 queue()传递函数数组的形式不能用于将多个函数添加到队列的末尾，因为任何现有的队列函数都将被删除。（为了添加多个函数到队列中，我们需要获

取函数数组，合并新的函数，然后将修改后的数组添加到队列中。）

2. 执行队列中的函数

好，现在可以添加函数到队列中来执行。这并不是很有用，除非能够以某种方法使函数的执行真正发生。下面来看下 dequeue()方法。

方法语法：dequeue

dequeue(name)

删除匹配集中每个元素的命名队列中的第一个函数并为每个元素执行此函数

参数

name （字符串）队列的名称，该队列中的第一个函数将会被删除并执行。如果省略，则默认为特效队列名称 fx

返回值

包装集

当调用 dequeue()时，会执行包装集中每个元素的命名队列中的第一个函数，并将当前元素设置为调用函数的上下文（this）。

下面考虑代码清单 5-8 中的代码。

代码清单 5-8 在多个元素上使函数入队和出队

```
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="../styles/core.css" />
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript" src="console.js"></script>
    <script type="text/javascript">
      $(function() {

        $('img').queue('chain',
         function(){ say('First: ' + $(this).attr('alt')); });
        $('img').queue('chain',
          function(){ say('Second: ' + $(this).attr('alt')); });
        $('img').queue('chain',
          function(){ say('Third: ' + $(this).attr('alt')); });
        $('img').queue('chain',
          function(){ say('Fourth: ' + $(this).attr('alt')); });

        $('button').click(function(){
          $('img').dequeue('chain');
        });

      });
    </script>
  </head>

  <body>
```

❶ 创建 4 个队列函数

❷ 每次单击使一个函数出队

```
    <div>
      <img src="earth.png" alt="Earth"/>
      <img src="moon.png" alt="Moon"/>
    </div>

    <button type="button" class="green90x24">Dequeue</button>

    <div id="console"></div>

  </body>
</html>
```

本例中（可以在文件 chapter5/queue.html 中找到），我们分别在两张图片上创建了名为 chain 的队列。在每个队列中放置了 4 个函数❶，通过序号来找出每个函数并且输出任何作为函数上下文的 DOM 元素的 alt 属性。通过这种方式可以知道哪个函数正在被执行以及该函数来自哪个元素的队列。

单击 Dequeue 按钮，按钮的单击处理器❷就会执行一次 dequeue()方法。

下面再单击按钮一次，观察控制台的消息，如图 5-8 所示。

5

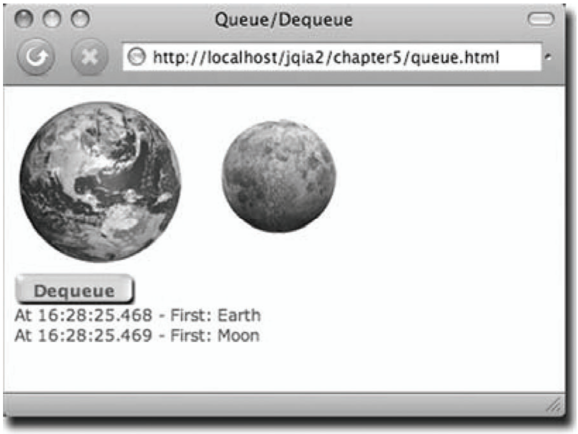


图 5-8 单击 Dequeue 按钮，触发队列中的单个函数实例并在每个创建队列的图片上执行一次

可以看到，添加到图片 chain 队列的第一个函数被触发了两次：第一次为 Earth 图片，第二次为 Moon 图片。

多次单击按钮会导致每次从队列中删除一个随后的函数，并且执行被删除的函数直到队列为空；在这之后，调用 dequeue()不会有任何效果。

在本例中，函数的出队是手动控制的——需要单击按钮 4 次（导致调用 dequeue()方法 4 次）才能使所有 4 个函数都被执行。我们可能经常希望执行整个集合中的所有队列函数。对于这种情况，常用的做法是在队列函数内部调用 dequeue()方法以便触发下一个队列函数的执行（换句话说，创建一条到下一个队列函数的链接）。

考虑对代码清单 5-8 中的代码做如下改动：

```
$('img').queue('chain',
 function(){
   say('First: ' + $(this).attr('alt'));
   $(this).dequeue('chain');
 });
$('img').queue('chain',
  function(){
    say('Second: ' + $(this).attr('alt'));
    $(this).dequeue('chain');
  });
$('img').queue('chain',
  function(){
    say('Third: ' + $(this).attr('alt'));
    $(this).dequeue('chain');
  });
$('img').queue('chain',
  function(){
    say('Fourth: ' + $(this).attr('alt'));
    $(this).dequeue('chain');
  });
```

我们已经在示例文件 chapter5/queue.2.html 中完成了这个改动。在浏览器中打开这个页面，然后单击 Dequeue 按钮。注意观察现在单个单击是如何执行整个队列函数链的。

3. 清除没有执行的队列函数

如果想从队列中删除那些没有执行的队列函数，那么可以使用 `clearQueue()`方法。

方法语法：**clearQueue**	
clearQueue(name)	
从命名队列中删除所有未执行的函数	
参数	
name	（字符串）队列的名称，该队列中未执行的函数将被删除。如果省略，则默认为特效队列名称 fx
返回值	
包装集	

和 `stop()`动画方法类似，`clearQueue()`可以在一般的队列函数上使用而不是仅仅用于动画特效。

4. 延迟队列函数

另一个适用于队列的操作是添加队列函数执行之间的延迟。`delay()`方法能够做到这一点。

方法语法：delay

delay(duration, name)

为命名队列中所有未执行的函数添加延迟[①]

参数

duration	（数字|字符串）延迟时间，可以是若干毫秒，也可以是字符串 fast 或者 slow 两者之一，分别表示 200 毫秒和 600 毫秒
name	（字符串）队列的名称，该队列中没有执行的函数将被删除。如果省略，则默认为特效队列名称 fx

返回值

包装集

在继续学习之前还有一件有关队列函数的事情需要讨论……

5.4.3 插入函数到特效队列

之前曾提到，jQuery 为了实现动画在内部使用一个名为 fx 的队列来对必要的函数进行排队。如果想向这个队列添加自定义函数，以便在一系列的特效队列中插入一些行为，该怎么办呢？现在知道了有关排队的方法，我们就可以做到了！

回想下前面代码清单 5-7 的例子，通过 4 个动画使 Moon 绕 Earth 转动。想象一下，要在第 2 个动画（使图片向下移动的动画）之后将 Moon 图片的背景色变为黑色。如果只是在第 2 个和第 3 个动画之间调用 css()方法，如下所示：

```
$("img[alt='moon']").animate({left:'+=256'},2500);
$("img[alt='moon']").animate({top:'+=256'},2500);
$("img[alt='moon']").css({'backgroundColor':'black'});
$("img[alt='moon']").animate({left:'-=256'},2500);
$("img[alt='moon']").animate({top:'-=256'},2500);
```

效果将非常不理想，因为这将会立即改变背景色，甚至可能在第 1 个动画开始之前。

相反，考虑如下代码：

```
$("img[alt='moon']").animate({left:'+=256'},2500);
$("img[alt='moon']").animate({top:'+=256'},2500);
$("img[alt='moon']").queue('fx',
  function(){
    $(this).css({'backgroundColor':'black'});
    $(this).dequeue('fx');
   }
);
$("img[alt='moon']").animate({left:'-=256'},2500);
$("img[alt='moon']").animate({top:'-=256'},2500);
```

① 这个说法不正确，delay 只能延迟队列中下一个函数的执行，而不会延迟所有未执行的函数。例如 `$("img").slideUp(300).delay(1000).fadeIn(500).slideUp(500);`，其中最后一个 slideUp 会在 fadeIn 完成之后立即执行。

这里使用 queue()方法将 css()方法的调用放在 fx 队列的一个函数中。(可以省略队列的名称，因为 fx 是默认的，但是为了清晰起见显式地声明了它。)这把改变颜色的函数置于特效队列中作为函数链的一部分，随着动画的进行，在第 2 个和第 3 个动画之间将调用该函数。

注意，我们在调用 css()方法之后，又调用了 fx 队列的 dequeue()方法。为了保持动画队列的连续性，这是绝对必要的。此时如果不调用 dequeue()会导致动画停顿下来，因为没有引发链中的下一个函数执行的因素。未执行的动画只是呆在特效队列中，直到某件事情导致一个出队操作才会引发函数执行，也可以卸载页面，这将丢弃所有的东西。

如果你想亲眼看到运行过程，可以在浏览器中载入页面文件 chapter5/revolutions.2.html，然后单击页面上的按钮。

当我们想要连续地执行函数时，函数队列就派上用场了，这样就不用在异步的回调中嵌套大量复杂的函数；你可能会联想到，函数队列在 Ajax 中也有用武之地。

不过那就是另一章的内容了。

5.5　小结

本章不仅介绍了直接使用的 jQuery 动画特效，还介绍了允许创建自定义动画的 animate()方法。

当不带参数调用 show()和 hide()方法时，会立即从显示器上显示或隐藏元素，没有任何动画。可以通过向这些方法传入控制动画速度的参数以及提供一个可选的、在动画完成时执行的回调来执行显示或隐藏元素的动画版本。toggle()方法用来切换元素的隐藏或显示状态。

另外一组包装器方法（fadeOut()和 fadeIn()）通过调整元素的不透明度来隐藏或显示元素，从而在显示器上删除或显示元素。第三种方法（fadeTo()）以动画方式改变包装元素的不透明度，但不会从显示器上删除元素[1]。

最后一组内置的 3 个特效动画通过调整包装元素的垂直高度来隐藏或显示它们：slideUp()、slideDown()以及 slideToggle()。

jQuery 提供了 animate()方法以便我们创建自定义动画。可以使用这个方法对任何接受数字值的 CSS 样式属性应用动画，最常用的是元素的不透明度、位置和尺寸。我们研究编写了一些自定义动画，用于从页面上以新颖、流行的方式删除元素。

我们也了解到 jQuery 如何将动画排队以便串行执行，以及如何使用 jQuery 的队列方法向特效队列或者自定义的队列中添加自定义函数。

在探讨如何编写自定义动画时，我们为这些自定义动画特效编写了页面中内联的 JavaScript 代码。一个更加常见和有用的方法是将自定义动画打包成自定义的 jQuery 方法。在第 7 章将会学习如何做到这一点，我们鼓励你在阅读完第 7 章后重温这些特效。请重新封装已在本章中开发的自定义特效以及任何所能想到的特效，这将会是个很棒的跟进练习。

但在编写自己的 jQuery 扩展之前，先来看一些高级的 jQuery 函数和标志，它们对于一般的任务和扩展功能都非常有用。

---

① 不从显示器上删除元素，指的是不将元素的 display 样式属性设置为 none。

第 6 章

DOM 无关的jQuery实用函数

6

本章内容

- ❑ jQuery 的浏览器支持信息
- ❑ 使用 jQuery 与其他的库
- ❑ 操作数组的函数
- ❑ 扩展与合并对象
- ❑ 动态加载新的脚本
- ❑ 更多……

目前为止，我们已经用了几章来介绍 jQuery 方法，这些方法作用于通过$()函数包装的 DOM 元素集合。你可能还记得，早在第 1 章我们就介绍了实用函数的概念——定义在 jQuery/$命名空间下不操作包装集的函数。这些函数可以看作是定义在$实例而不是 window 实例上的顶级函数，从而将它们排除在全局命名空间之外。

一般来说，这些实用函数要么操作除 DOM 元素（毕竟，这是包装器方法的管辖范围）以外的 Java Script 对象，要么执行一些对象无关的操作（例如 Ajax 请求）。

除了函数，jQuery 还提供了一些定义在 jQuery/$命名空间中的有用标志。

你可能会奇怪为什么要等到这一章才介绍这些函数和标志。有如下两个理由。

- ❑ 我们想引导你从 jQuery 包装器方法的角度来思考问题，而不是依靠底层的操作，虽然底层操作你可能更加熟悉，但不如使用 jQuery 包装器编写代码来得高效和简单。
- ❑ 因为包装器方法涵盖了非常多的方法，它们可以操作页面上 DOM 元素，所以这些底层函数在用来编写这些方法本身时最为有用（以及其他扩展），而不是用在编写页面级别的代码里。（我们将在第 7 章学习如何编写自己的 jQuery 插件。）

在本章中，我们终于将正式地介绍大部分$级别的实用函数以及一些有用的标志。涉及 Ajax 的实用函数将会推迟到第 8 章探讨，其中会专门介绍 jQuery 的 Ajax 功能。

下面先从提到的那些标志开始。

6.1 使用 jQuery 标志

jQuery 是通过定义在$上的属性来为页面开发者和插件开发者提供一些信息，而不是通过方

法或者函数提供的。其中的很多标志用来检测当前浏览器的功能，而其他标志则用来帮助我们在页面全局级别的层次上控制 jQuery 的行为。

公共用途的 jQuery 标志如下：

- `$.fx.off`——启用或者禁用特效；
- `$.support`——所支持特征的详细信息；
- `$.browser`——公开浏览器的细节（官方废弃）。

先来看下 jQuery 如何禁用动画。

6.1.1　禁用动画

有时，我们可能希望在包含各种动画效果的页面上有条件地禁用动画。这么做可能是因为已经检测到当前设备或者平台不能很好地处理动画，或者由于可访问性的原因。

无论哪种情况，都不需要编写两个页面，一个使用动画而另一个不使用动画。当检测到当前处于不支持动画的环境时，设置`$.fx.off`为`true`即可。

这不会废除已经在页面上使用的任何特效，只是简单地禁用了这些动画特效。例如，消退特效会立即显示或隐藏元素，没有过渡动画。

类似地，调用`animate()`方法将会设置 CSS 属性为指定的最终值，跳过动画过程。

比如在某些不能正确支持动画的移动设备或者浏览器中，就可能会用到这个标志。在那种情况下，你可能想要关闭动画以便核心功能正常运行。

`$.fx.off`是一个可读/写的标志，其余的预定义标志是只读的。下面来看下浏览器的用户代理（User Agent）提供的有关环境信息的标志。

6.1.2　检测用户代理支持

值得高兴和感激的是，迄今为止已研究的 jQuery 方法屏蔽了浏览器的差异，即便是传统的问题区，例如事件处理也是如此。但是，当我们自己写这些方法（或者其他扩展）的时候，可能需要考虑浏览器操作方式的差异，以便让使用扩展的用户不用关心这些差异。

在深入了解 jQuery 如何在实现这个功能之前，先来探讨下浏览器检测的整个概念。

1. 为什么浏览器检测是可憎的

好吧，或许“可憎的”这个词有点过了，但是除非绝对必要，否则应该尽量避免使用浏览器检测技术。

浏览器检测初看起来似乎是处理浏览器差异的合乎逻辑的方式。毕竟，很容易这么说，“我知道浏览器 X 的功能集，因此面向浏览器的测试绝对行得通，不是吗？”但是浏览器检测却充满了陷阱和问题。

反对这种技术的主要论点之一是浏览器种类繁多，再加上同一浏览器的不同版本支持的级别不同，使得这一技术无法解决可扩展性问题。

你可能会想，“好，我只需要测试 IE 和 Firefox 浏览器。”但是怎能排除不断增长的 Safari 用户呢？还有 Opera 和 Google 的 Chrome 浏览器呢？此外，还有一些小众的但并非无足轻重的浏览

器，它们和更加流行的浏览器分享功能配置信息。例如，Camino 是一款与 Firefox 使用相同的技术并拥有 Mac 友好的用户界面的浏览器。OmniWeb 则使用与 Safari 和 Chrome 相同的渲染引擎。

没有必要排除对这些浏览器的支持，但是对它们进行测试却是*相当痛苦*的。而这还没有考虑不同版本之间的差异——例如 IE 6、IE 7 和 IE 8。

另一个原因是，如果对某个特定的浏览器进行测试，但是未来发布的版本修正了该问题，那我们的代码实际上可能就不工作了。jQuery 解决这一问题的替代方法（将在下一节中讨论）是鼓励浏览器厂商来修复这些 jQuery 已经解决了的问题。

最后一个反对浏览器检测（有时也被称为*嗅探*）的论点是，目前越来越难知道“谁是谁”了。

浏览器通过设置一个称为*用户代理*字符串的请求头（request header）来标识自己。解析这个字符串就已经让人望而却步了。况且现在很多浏览器允许用户伪造（spoof）这个字符串，因此在克服所有困难解析了这个字符串之后，也不能相信它就是真实的结果。

一个名为 `navigator` 的 JavaScript 对象包含了用户代理的一部分信息，但是它也存在浏览器差异[①]。我们几乎是为了进行浏览器检测而必须进行浏览器检测！

停止这疯狂的行为！

浏览器检测的特征如下。

- 不精确——偶尔会意外地限制一些浏览器，而这些浏览器本来可以正常运行代码。
- 不可扩展——为了梳理事情而导致大量嵌套的 `if` 和 `if-else` 语句。
- 不准确——因为用户可以伪造用户代理信息。

很明显，只要可能就应该避免使用浏览器检测这个方法。

但是能够做些什么来代替呢？

2. 浏览器检测的替代方案是什么

如果仔细想一下，就可以知道我们对用户使用的是什么浏览器并不*真正*感兴趣，难道不是吗？考虑浏览器检测的唯一原因，是因为需要知道可以使用浏览器的哪些功能和特征。我们真正想得到的是浏览器的*功能和特征*，而浏览器检测不过是试图确定这些功能和特征的笨拙方式。

因此为什么不直接搞清楚这些特性，而是试图通过浏览器识别来推断它们呢？被广泛称为*特征检测*的技术允许代码基于特定的对象、属性，甚至是方法是否存在来进行分支。

作为示例，回想一下第 4 章的事件处理内容。记得有两个高级事件处理模型：W3C 标准 DOM 第 2 级事件模型和 IE 专有的事件模型。这两个模型都在 DOM 元素上定义方法以便创建监听器，但是它们使用不同的方法名称。标准模型定义的方法为 `addEventListener()`，而 IE 模型定义的方法为 `attachEvent()`。

使用浏览器检测，而且假设我们已经克服了确定当前使用的是什么浏览器（可能是正确的）的痛苦和烦躁，可以这样编写代码：

```
...
complex code to set flags: isIE, isFirefox, and isSafari
...
if (isIE) {
```

① 这里指的是不同浏览器中 `window.navigator.userAgent` 的字符串格式各不相同。

6

```
  element.attachEvent('onclick',someHandler);
}
else if (isFirefox || isSafari) {
  element.addEventListener('click',someHandler);
}
else {
  throw new Error('event handling not supported');
}
```

除了这个例子所掩盖的事实，也就是必然要使用复杂的代码来设置标志 isIE、isFirefox、isSafari 之外，我们也不能确定这些标志是否精确地描述了当前使用的浏览器。此外，这段代码在 Opera、Chrome、Camino、OmniWeb 或者在其他许多不知名的浏览器中使用时将会抛出错误，尽管这些浏览器可能完美地支持标准模型。

考虑这段代码的如下变体：

```
if (element.attachEvent) {
  element.attachEvent('onclick',someHandler);
}
else if (element.addEventListener) {
  element.addEventListener('click',someHandler);
}
else {
  throw new Error('event handling not supported');
}
```

这段代码没有进行大量复杂的、基本上不可靠的浏览器检测，但它支持所有的浏览器，只要这些浏览器支持两种对立事件模型中的任何一个即可。这好多了！

特征检测大大优于浏览器检测。特征检测更加可靠，并且不会意外地限制那些支持所测试功能的浏览器，只不过因为我们不了解该浏览器的特征或者不了解浏览器本身。在最近的 Web 应用中你考虑到 Google 的 Chrome 浏览器了吗？考虑到 iCab、Epiphany、Konqueror 了吗？[①]

注意　除非绝对必要，即使是特征检测也应该尽量避免。如果有一个跨浏览器的解决方案，那么应该优先考虑它而不是任何类型的代码分支。

尽管特征检测可能比浏览器检测更具有优势，但是它并非轻而易举。在页面中引入分支和类型检测依然是乏味和痛苦的，并且一些特征差异非常难于检测，需要繁琐的或者全面的复杂检查。jQuery 帮助我们执行这些检查，并提供一组标志用来检测我们可能关心的、最常见的用户代理特征。

3. jQuery 浏览器功能标志

浏览器功能标志是作为 jQuery 的$.support 对象的属性公开的。

表 6-1 总结了该对象的可用标志。

① iCab 是 Mac 下的浏览器，Konqueror 和 Epiphany 是 Linux 下的浏览器。

表6-1　**$.support**浏览器功能标志

标志属性	描　述
boxModel	如果用户代理按照标准兼容的盒模型渲染，则设置为true。这个标志直到文件就绪后才设置。更多有关盒模型问题的描述可以从以下网址获取：http://www.quirksmode.org/css/box.html和http://www.w3.org/TR/ RECCSS2/box.html
cssFloat	如果使用了标准的cssFloat属性来描述元素的style属性，则设置为true
hrefNormalized	如果获取元素的href特性得到的值和指定的值一模一样，则设置为true
htmlSerialize	如果浏览器对通过innerHTML插入DOM中的<link>元素所引用的样式表求值，则设置为true
leadingWhitespace	如果浏览器保留通过innerHTML插入文本的开头空白字符，则设置为true
noCloneEvent	如果浏览器在复制元素时不复制事件处理器，则设置为true
objectAll	如果向JavaScript的getElementsByTagName()方法传入“*”时该方法返回所有的后代元素，则设置为true
opacity	如果浏览器正确解析标准的CSS属性opacity，则设置为true
scriptEval	如果浏览器对通过appendChild()或createTextNode()方法插入到DOM中的<script>块求值，则设置为true
style	如果用来获取元素内联样式属性的特性名称是style则设置为true
tbody	如果通过innerHTML插入缺少<tbody>的表元素时浏览器不自动向表中添加此元素，则设置为true

表 6-2 显示了这些标志在不同浏览器中的值。

表6-2　**$.support**标志在各个浏览器中的值

标志属性	Gecko（Firefox、Camino等）	WebKit（Safari、OmniWeb、Chrome等）	Opera	IE
boxModel	true	true	true	quirks模式下为false，标准模式下为true
cssFloat	true	true	true	false
hrefNormalized	true	true	true	false
htmlSerialize	true	true	true	false
leadingWhitespace	true	true	true	false
noCloneEvent	true	true	true	false
objectAll	true	true	true	false
opacity	true	true	true	false
scriptEval	true	true	true	false
style	true	true	true	false
tbody	true	true	true	false

正如所料，这归根结底是 IE 浏览器和标准兼容的浏览器之间的差异。但是不要以为可以回到浏览器检测，那种方式会以失败告终，因为 IE 浏览器的未来版本可能会修正漏洞和差异。记住，其他浏览器也可能会在无意中引入问题和差异。

当需要进行功能判断时特征检测总是优于浏览器检测，但是特征检测未必总能解决问题。在一些很少见的情况下，我们需要求助于特定的浏览器判断，而这只能通过浏览器检测来完成（很快就会看到一个示例）。对于这些情况，jQuery 提供了一组标志来进行直接的浏览器检测。

6.1.3　浏览器检测标志

在只能使用浏览器检测的情况下，可以使用 jQuery 提供的一组标志来进行分支。这些标志在加载库时被创建，这使得它们在任何就绪处理器执行前就已经可用了，并被定义为通过 `$.browser` 引用的对象实例的属性。

注意，尽管这些标志在 jQuery1.3 以及以后的版本中依然存在，但却被视为已废弃，这意味着可以在将来任何 jQuery 发布版中删除，因此在使用这些标志时应该考虑到这一点。在浏览器发展停滞的时期这些标志可能会更有用，但是现在处于浏览器蓬勃发展的时代，功能支持标志更有意义并且很可能会存在一段时间。

事实上，当核心支持标志没有提供所需的值时，建议你创建一个新的自定义标志。不过我们很快就会探讨这方面的内容。

表 6-3 描述了浏览器支持标志。

注意这些标志不会试图识别当前使用的具体的浏览器。jQuery 根据浏览器所属的家族对用户代理进行分类，通常根据使用的渲染引擎进行判定。属于同一家族的浏览器具有相同的几组性质，因此没必要识别具体的浏览器。

表6-3　**`$.browser`用户代理检测标志**

标志属性	描　述
`msie`	如果用户代理被识别为任意版本的IE浏览器，则设置为`true`
`mozilla`	如果用户代理被识别为任意基于Mozilla的浏览器，则设置为`true`。这类浏览器包括Firefox和Camino
`safari`	如果用户代理被识别为任意基于WebKit的浏览器，则设置为`true`。比如Safari、Chrome以及OmniWeb
`opera`	如果用户代理被识别为Opera浏览器，则设置为`true`
`version`	设置为浏览器渲染引擎的版本号

绝大多数常用的现代浏览器都属于这四个浏览器家族之一，包括 Google 的 Chrome 浏览器，由于其使用的是 WebKit 引擎，所以返回的 `safari` 标志为 `true`。

`version` 属性特别值得关注，因为它可能不像我们想像的那样方便。这个属性的值不是浏览器的版本号（刚开始我们可能这样认为）而是浏览器渲染引擎的版本号。例如，当在 Firefox3.6 下运行时，报告的版本号是 1.9.2——Gecko 渲染引擎的版本号。用这个值来区分 IE 浏览器的版本是很方便的，因为 IE 浏览器的渲染引擎版本号和浏览器版本号是匹配的。

前面曾提到，有时我们不能指望特征检测而必须借助于浏览器检测。这种情况的一个示例是，浏览器之间的差异并不是它们使用不同的对象类或者不同的方法，而是传入方法的参数在不同浏览器下的实现有不同的解释。在这种情况下，就没有用于检测的对象或者特征了。

注意 即使在这些情况下，仍然可以通过在页面的隐藏区域内进行操作来设置特征标志（正如 jQuery 设置它的一些特征标志那样）。但是，我们很少在页面上看到这种 jQuery 之外技术。

以<select>元素的 add()方法为例。W3C 对其定义如下（想参考规范的开发者请移步到这里：http://www.w3.org/TR/DOM-Level-2-HTML/html.html#ID-14493106）：

```
selectElement.add(element,before)
```

对于这个方法，第一个参数找出需要添加到<select>元素的<option>或者<optgroup>元素，第二个参数找出已有的<option>（或者<optgroup>）元素，新元被放置于该元素之前。在标准兼容的浏览器中，第二个参数是对指定的现有元素的*引用*，而在 IE 浏览器中，它是现有元素的*序号索引*。

因为没有办法进行特征检测来决定是应该传递对象引用还是整数值（缺少尝试，如前所述），所以要借助于浏览器检测，如下面的例子所示：

```
var select = $('#aSelectElement')[0];
select.add(
  new Option('Two and \u00BD','2.5'), $.browser.msie ? 2 : select.options[2]
);
```

在这段代码中，我们对$.browser.msie 进行简单的测试来决定是应该传递序号值 2 还是<select>元素中第三项的引用。

然而，jQuery 团队建议我们不要在代码中直接使用浏览器检测。推荐的方法是创建一个自定义的支持标志来把浏览器检测提取出来。这样一来，一旦浏览器支持标记消失了，仅仅需要寻找另一种方式来在另一个地方设置标记即可，从而避免修改代码。

例如，在我们自己的 JavaScript 库代码中，可以这样写：

```
$.support.useIntForSelectAdds = $.browser.msie;
```

还可以在代码中使用这个标记。如果浏览器检测标志被删除，只需要改变库代码，所有使用这个自定义标志的代码都不受影响。

现在，离开标志的世界，来看看 jQuery 提供的实用函数。

6.2 jQuery 与其他库并存

早在第 1 章我就介绍了 jQuery 团队提供了的一个体贴周到的方法，用来在同一个页面上轻松地使用 jQuery 和其他库。

通常，当在同一个页面上使用 jQuery 和其他库时，全局名称$的定义是最大的争论和冲突的焦点。众所周知，jQuery 使用$作为 jQuery 名称的别名，并将其用于 jQuery 公开的每一个功能。但是其他库，最著名的就是 Prototype，也使用$名称。

jQuery 提供了$.noConflict()实用函数用来放弃对$标识符的占用，以便其他库使用它，该函数的语法如下。

函数语法：`$.noConflict`

`$.noConflict(jqueryToo)`

将标识符$的控制权归还给其他库，允许在页面上混合使用 jQuery 与其他库。一旦执行了该函数，必须使用 `jQuery` 标识符而不是$标识符来调用 jQuery 的功能

你也可以放弃 `jQuery` 标识符（可选）

应该在包含了 `jQuery` 之后，但尚未包含冲突库之前调用这个方法

参数

`jqueryToo`	（布尔）如果提供了该参数并且设置其为 `true`，则会一并放弃$和 `jQuery` 标识符

返回值

jQuery

尽管使用的是 `jQuery` 标识符，但是因为$是 `jQuery` 的别名，所以在应用`$.noConflict()`之后所有 jQuery 的功能依然可用。作为对失去心爱的$符号的补偿，我们可以定义更短的、但没有冲突的 `jQuery` 别名，例如：

```
var $j = jQuery;
```

另一个常见的习惯用法是创建一个作用域环境，在该环境中$标识符指向 `jQuery` 对象。在扩展 jQuery 的时候这是个常用技巧，特别是对于插件作者来说，他们不可能对页面开发者是否已经调用`$.noConflict()`作出任何假设，当然也不能自行调用此函数以免破坏页面开发者的意愿。

这个习惯用法如下：

```
(function($) { /* 这里是函数主体 */ })(jQuery);
```

如果这个符号让你感到头疼，别担心！虽然第一次看起来很奇怪，但它其实非常简洁明了。

下面来剖析这个习惯用法的第一部分：

```
(function($) { /* 这里是函数主体 */ })
```

这部分声明了一个函数并用圆括号括起来，由此生成一个表达式，这个表达式的结果是对一个匿名函数的引用。这个函数期望传入单个参数并将其命名为$。在函数主体中，可以通过$标识符来引用任何传递给这个函数的东西。因为参数声明优先于全局作用域中任何类似的命名标识符，所以任何在函数外定义的$值在函数内都会被传入的参数所替代。

这个习惯用法的第二部分：

```
(jQuery)
```

在匿名函数上执行函数调用，将 `jQuery` 对象作为参数传递。

结果，在函数外部不管$标识符是否已经在 Prototype 或其他库中定义，在函数体内它总是指向 `jQuery` 对象。很漂亮，对吧？

当使用这个技巧时，外部声明的$在函数体内是不可用的。

除了在第 1 章介绍的方法外，这个习惯用法的一个变体也经常用于声明就绪处理函数，从而形成了第三种语法。考虑如下代码：

```
jQuery(function($) {
  alert("I'm ready!");
});
```

和第 1 章介绍的一样，通过将一个函数作为参数传入 jQuery 函数来声明就绪处理器。但是这次，我们使用$标识符声明传入就绪处理器的单个参数。因为 jQuery 总是将 jQuery 引用作为第一个也是唯一的参数传入就绪处理器，所以确保了在就绪处理器内$名称指向 jQuery，而无论在处理器外部$的定义是什么。

接下来通过一个简单的测试来证明这一点。测试的第一部分，先来看看代码清单 6-1 中的 HTML 文档（可以从文件 chapter6/ready.handler.test.1.html 获取）。

代码清单 6-1　就绪处理器测试

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hi!</title>
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
    <script type="text/javascript">
      var $ = 'Hi!';
      jQuery(function(){
        alert('$ = '+ $);
      });
    </script>
  </head>
  <body></body>
</html>
```

❶ 使用自定义值覆盖$名称

❷ 声明就绪处理器

在这个例子中，我们引入 jQuery 库。众所周知，jQuery 库定义了全局名称 jQuery 和别名$。然后将全局变量$重定义为字符串值❶，覆盖了 jQuery 的定义。为简单起见，本例中将$替换为一个字符串值，但是也可以通过包含另外一个库（例如 Prototype）来重定义$。

然后定义了就绪处理器❷，它的唯一作用是弹出一个显示$值的警告对话框。

当页面加载时，将会看到一个警告对话框，如图 6-1 所示。

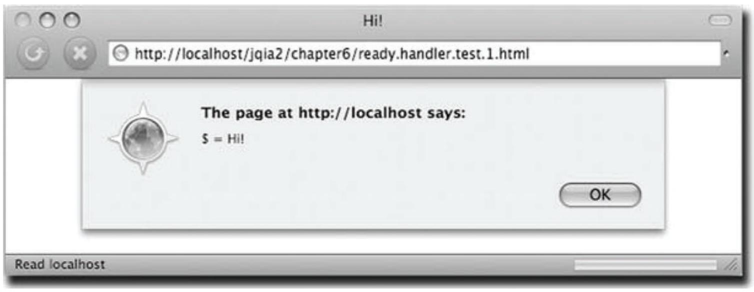


图 6-1　由于对$的重定义已生效，因此在就绪处理器中它的值是“Hi!”

注意，在就绪处理器作用域中，全局变量$的值被重新定义为字符串赋值语句的结果。如果想在处理器中使用 jQuery 定义的$值，那会非常失望。

现在对这个示例文档做个修改。下面的代码只显示文档已修改的部分；小的改动以粗体突出显示。（可以从 chapter6/ready.handler.test.2.html 获取完整的页面。）

```
<script type="text/javascript">
  var $ = 'Hi!';
  jQuery(function($){
    alert('$ = '+ $);
  });
</script>
```

我们所做的唯一改变就是为就绪处理器函数添加了一个名为$的参数。加载这个修改后的版本，会看到完全不一样的结果，如图 6-2 所示。

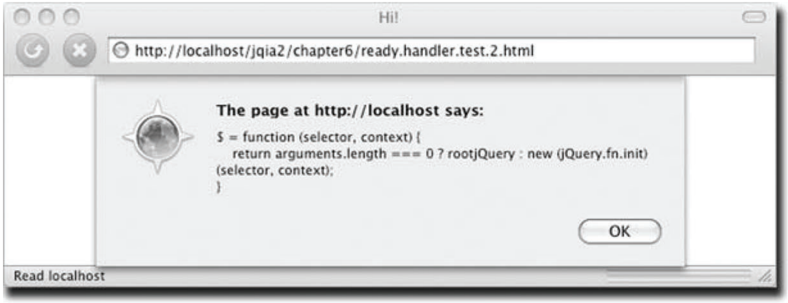


图 6-2　现在警告对话框显示的是 jQuery 版本的$，因为在函数内它的定义被强制生效了

这可能和事先预料的结果不完全一致，但是快速浏览一下 jQuery 的源代码就会发现，因为我们在 jQuery 函数中将就绪处理器的第一个参数声明为$，所以$标识符指向被 jQuery 库当作唯一的参数传入所有就绪处理器的 jQuey 函数（因此警告对话框显示的是 jQuery 函数的定义）。

当编写可重用的组件并且这些组件可能会用于已经使用了$.noConflict()的页面时，最好对$的定义采取这种预防措施。

还有大量的 jQuery 实用函数可以用来操作 JavaScript 对象。下面来仔细学习这些函数。

6.3　操作 JavaScript 对象和集合

作为实用函数来实现的多数 jQuery 功能被设计用来操作 JavaScript 对象而非 DOM 元素。一般来说，任何在 DOM 上操作的功能都是作为 jQuery 包装器方法提供的。尽管部分实用函数可以用来操作 DOM 元素（毕竟它们也是 JavaScript 对象），但是实用函数的关注点不是 DOM。

这些函数涵盖了很多内容，从简单的字符串操作、类型检测到复杂的集合过滤、序列化表单值，甚至涵盖了通过属性合并来实现对象继承的一种形式。

先从最基础的开始学习。

6.3.1　修剪字符串

真是不可思议，JavaScript 的 String 类型居然没有一种方法来删除一个字符串实例开头和结尾空白字符。在大多数其他语言中，这么基本的功能通常是 String 类的一部分，但是 JavaScript 却缺少这个有用的功能，让人很费解。

然而修剪字符串是很多 JavaScript 应用中的常见需求；一个典型示例是在表单数据验证阶段。由于空白字符在屏幕上是不可见的（因而得名"空白"），因此用户很容易在文本框或文本区中的有效数据项前后不小心输入多余的空白字符。在验证过程中，我们希望悄悄地从数据中删除这些空白字符，而不是提醒用户他们输入的数据存在一些不可见的错误。

为了帮助我们，jQuery 定义了$.trim()函数，语法如下所示。

函数语法：$.trim

$.trim(value)

删除传入的字符串开头和结尾处的空白字符，并返回修改后的结果

这个函数中的空白字符被定义为匹配 JavaScript 正则表达式\s 的任意字符，不仅匹配空白字符，而且匹配换页、换行、回车、制表，以及垂直制表符，还包括 Unicode 字符\u00A0

参数

value　（字符串）需要修剪的字符串值。原始字符串值不会改变

返回值

修剪后的字符串

使用这个函数原地修剪一个文本字段值的示例如下：

```
$('#someField').val($.trim($('#someField').val()));
```

注意，这个函数不会检查传入的参数来确保它是一个 String 值，因此如果向这个函数传入其他类型的值，我们将得到 undefined 或者令人遗憾的结果（可能是一个 JavaScript 错误）。

下面来看一些在数组和其他对象上操作的函数。

6.3.2　遍历属性和集合

当我们拥有其他组件组成的非标量值时，经常需要遍历其中包含的项目。不管容器元素是一个 JavaScript 数组（包含任意个数的其他 Javascript 值，含数组在内）还是 JavaScript 对象的实例，JavaScript 语言都提供了对其进行遍历的方法。对于数组，使用 for 循环来遍历其中的元素；对于对象，则使用 for-in 循环来遍历其中的属性。

各编写一个示例，如下所示：

```
var anArray = ['one','two','three'];
for (var n = 0; n < anArray.length; n++) {
  //do something here
```

```
}

var anObject = {one:1, two:2, three:3};
for (var p in anObject) {
  //do something here
}
```

上述代码非常简单，但是有些人可能认为语法有点罗嗦和复杂——这也是对 for 循环的常见批评。我们知道，jQuery 定义了 each()方法来操作 DOM 元素的包装集，允许我们轻松地遍历集合中的元素而无需使用麻烦的 for 循环语法。对于普通的数组和对象，jQuery 提供了类似的名为$.each()的实用函数。

使用这个函数的好处是，无论是遍历数组中的项目还是对象中的属性，使用的语法都是一样的。

函数语法：$.each

$.each(container, callback)

遍历传入的容器中的每一项，并为每一项调用传入的回调函数

参数

container	（数组|对象）一个数组，其每一项都将被遍历；或者一个对象，其每一个属性都将被遍历
callback	（函数）为容器中的每个元素调用的回调函数。如果容器是一个数组，则为每一个数组项调用回调函数；如果容器是一个对象，则为对象的每一个属性调用回调函数 回调函数的第一个参数是数组元素的下标或对象属性的名称。第二个参数是数组项或者属性值。将传入的第二个参数的值设置为调用函数的上下文（this）

返回值

容器对象

这种统一的语法使用相同的格式遍历数组或对象。使用这个函数编写上面的示例，如下所示：

```
var anArray = ['one','two','three'];
$.each(anArray,function(n,value) {
  //do something here
});

var anObject = {one:1, two:2, three:3};
$.each(anObject,function(name,value) {
  //do something here
});
```

在选择语法时虽然使用内联函数的$.each()似乎是一个半斤八两的解决方案，但是这个函数可以很容易地编写可重用的迭代器，或者为了使代码清晰可以很容易地将循环体提取到另外一个函数中，如下所示：

```
$.each(anArray,someComplexFunction);
```

注意，当遍历一个数组或者对象时，可以使迭代器函数返回 false 以便跳出循环。

注意 你可能还记得，也可以使用 each()方法来遍历数组[①]，但是相比 each()方法，$.each()函数有略微的性能优势。记住，如果你非常关心性能问题，那就需要使用老式的 for 循环来获得最佳的性能。

有时，我们需要遍历数组来选择一些元素并将其添加到一个新的数组中。尽管可以使用$.each()来达到目的，但是 jQuery 使这一切变得更加简单，下面就来看下吧。

6.3.3 筛选数组

对于需要频繁处理大量数据的应用来说，遍历数组以查找匹配一定标准的元素是常见需求。我们可能希望筛选数据，查找高于或者低于特定临界值，或者可能是匹配一定模式的数据。对于任何这种类型的筛选操作，jQuery 提供了$.grep()实用函数。

$.grep()函数的名称可能会让我们以为它使用正则表达式，就像其在 UNIX 下的同名命令 grep 那样。但是$.grep()实用函数使用的筛选标准不是正则表达式，而是由调用者提供的回调函数，用来决定数据值是否应该包含在结果值集合中。jQuery 不会阻止回调函数使用正则表达式来完成任务，但是正则表达式的使用不是自动完成的。

这个函数的语法如下所示。

函数语法：$.grep

$.grep(array, callback, invert)

遍历传入的数组，为每个元素调用回调函数。回调函数的返回值决定是否应该将当前值收集到一个新数组中，这个新数组将作为$.grep()函数的返回值。如果 invert 参数被省略或者为 false，回调函数返回 true 将导致数据被收集。如果 invert 设置为 true，回调函数返回 false 将导致数据被收集

原始数组不受影响

参数

array	（数组）需要遍历的数组，它的数据值将会被检查以确定是否应该收集到新数组中。这个操作不会以任何方式修改数组
callback	（函数）一个函数，它的返回值决定是否应该收集当前数据值。返回 true 则导致当前值被收集，除非 invert 参数的值设为 true，这种情况下会发生相反的事情 这个函数接受两个参数[②]：当前的数据值和其在原始数组中的下标
invert	（布尔）如果指定为 true，则反转函数的正常操作

返回值

由收集的值组成的数组

① 这里指的是 2.3.4 节中使用 each()方法遍历包装集中的元素。

② 注意，这个函数的参数顺序和$.each()回调函数的参数顺序刚好相反。

比方说，要筛选数组中所有大于 100 的值。可以通过如下语句实现：

```
var bigNumbers = $.grep(originalArray,function(value) {
                   return value > 100;
                 });
```

传入`$.grep()`的回调函数可以进行任何处理，以决定是否应该包含当前值。这个决定可能像上述示例一样简单，也可能像向服务器发出同步的 Ajax 请求（有一定的性能损失）以决定是否应该包含或排除当前值。

虽然`$.grep()`函数不直接使用正则表达式（别在意它的名称），但是 JavaScript 正则表达式是一个强大的工具，用来在回调函数中决定是否应该在结果数组中包含某些值。考虑这样一个场景，我们有一个数组并且希望标识出不匹配美国邮政编码（也被称为 Zip 编码）格式的任何值。

美国的邮政编码由 5 个十进制数组成，后面跟着一个破折号和另外 4 个十进制数（可选）。匹配这个模式的正则表达式是`/^\d{5}(-\d{4})?$/`，因此可以使用如下代码从源数组中过滤那些不符合标准的数据项：

```
var badZips = $.grep(
                  originalArray,
                  function(value) {
                    return value.match(/^\d{5}(-\d{4})?$/) != null;
                  },
                  true);
```

注意，这个示例使用了 `String` 类的 `match()`方法来决定一个值是否匹配模式，并且指定`$.grep()`的 `invert` 参数为 `true` 来排除任何匹配模式的值。

获取数组的数据子集并不是在数组上进行的唯一操作。下面来看看 jQuery 提供的另外一个面向数组的函数。

6.3.4　转换数组

数据并不总是以我们需要的格式存在。在以数据为中心的 Web 应用中，另一个常见操作是将一组值转换为另一组值。虽然编写 `for` 循环从一个数组创建另一个数组是一件简单的事情，但是 jQuery 的`$.map` 实用函数让这一任务更加简单。

函数语法：`$.map`

`$.map(array, callback)`

遍历传入的数组，为数组的每一项调用回调函数，并将函数调用的返回值收集到一个新的数组中

参数

`array`	（数组）一个数组，它的值将被转换为新数组中的值
`callback`	（函数）一个函数，它的返回值将被收集到一个新数组中，这个新数组作为`$.map()`函数的调用结果 这个函数接受两个参数：当前的数据值和其在原始数组中的下标

返回值

由收集的值组成的数组

来看一个展示如何调用$.map()函数的示例。

```
var oneBased = $.map([0,1,2,3,4],function(value){return value+1;});
```

这个语句将值从 0 开始的数组转换为相应的从 1 开始的数组。

需要注意的一个重要的行为是，如果函数返回 null 或者 undefined，那结果就不会被收集。在这种情况下，结果数组的长度将小于原始数组的长度，并且数组项之间一对一的顺序也不存在了。

下面看一个稍微复杂的示例。假设有一个期望表示数字值的字符串数组，也许是从表单字段收集来的，我们希望把这个字符串数组转换为相应的 Number 实例的数组。由于不能保证是否存在无效的数字字符串，因此需要事先采取预防措施。考虑如下代码：

```
var strings = ['1','2','3','4','S','6'];

var values = $.map(strings,function(value){
  var result = new Number(value);
  return isNaN(result) ? null : result;
});
```

先从一个字符串数组开始，其中的每一项都代表一个数字值。但是由于打字错误（或者用户输入错误）导致字母 S 替代了正确的数字 5。这段代码通过检查构造器创建的 Number 实例来判断从字符串到数字的转换是否成功。如果转换失败，返回的值将是常量 Number.NaN。但有趣的是，从定义上讲 Number.NaN 不等于任何其他的值，包括它自己！因此表达式 Number.NaN==Number.NaN 的结果是 false。

因为不能使用一个比较操作符测试 NaN（顺便提一下，它代表 Not a Number，即不是一个数字），JavaScript 提供了 isNaN()方法，用来测试从字符串到数字的转换结果。

在这个例子中，转换失败时返回 null，以确保结果数组中只包含有效的数字值同时也剔除了任何错误的值。如果想收集所有的值，可以让转换函数为那些错误的值返回 Number.NaN。

$.map()另一个有用的行为是，它可以优雅地处理从转换函数返回数组的情况，并将返回的值合并到结果数组中。考虑如下语句：

```
var characters = $.map(
  ['this','that','other thing'],
  function(value){return value.split('');}
);
```

这个语句将字符串数组转换为构成字符串的所有字符组成的数组。执行后变量 characters 的值如下所示：

```
['t','h','i','s','t','h','a','t','o','t','h','e','r',' ','t','h','i','n','g']
```

这是使用 String.split()方法完成的，当向其传递一个空字符串作为分隔符时它会返回包含字符串中字符的数组。这个数组作为转换函数的结果返回，并且被合并到结果数组中。

jQuery 对数组的支持远不止这些。还有一些较小的函数也会带来方便。

6.3.5　发现JavaScript数组的更多乐趣

你曾有过需要知道某个 JavaScript 数组是否包含特定值吗？甚至还想知道该值在数组中的位置吗？

如果是这样，你将会感激`$.inArray()`函数。

函数语法：`$.inArray`

`$.inArray(value, array)`

返回传入的值第一次出现时的下标

参数

value	（对象）需要在数组上搜索的值
array	（数组）将要被搜索的数组

返回值

该值在数组中第一次出现时的下标，如果没有查找到这个值，则返回−1

一个足以说明这个函数用法的示例：

```
var index = $.inArray(2,[1,2,3,4,5]);
```

这将导致下标值 1 被赋值给 index 变量。

还有另一个数组相关的有用的函数，这个函数可以从其他类似数组的对象创建 JavaScript 数组。“其他类似数组的对象？到底什么是类似数组的对象？”你可能会问。

jQuery 认为类似数组的对象就是拥有长度和下标项概念的任何对象。在处理 NodeList 对象时这个功能非常有用。考虑如下代码片段：

```
var images = document.getElementsByTagName("img");
```

这个语句将页面上所有图片组成的 NodeList 赋值给 images 变量。

处理 NodeList 有点痛苦[①]，因此将其转换为 JavaScript 数组要好很多。jQuery 的`$.makeArray`函数使得转换 NodeList 非常简单。

函数语法：`$.makeArray`

`$.makeArray(object)`

将传入的类似数组的对象转换为 JavaScript 数组

参数

object	（对象）需要被转换的类似数组的对象（比如 NodeList）

返回值

JavaScript 数组

该函数适合在较少使用 jQuery 的代码中使用，因为 jQuery 已经在其内部处理了这类事情。

① 这里指的是不能对 NodeList 使用数组的原型方法，比如 pop、push、slice 等。

该函数在下面情况中也派得上用场：处理`NodeList`对象时不使用 jQuery 来遍历 XML 文档，或者处理函数内的`arguments`实例（你可能会惊讶地发现，它不是标准的 JavaScript 数组）。

还有另一个很少使用到的函数，但在处理非 jQuery 创建的数组时会非常有用，它就是`$.unique()`函数。

函数语法：`$.unique`

`$.unique(array)`

向其传入 DOM 元素的数组，则返回由原始数组中不重复的元素组成的数组

参数

`array`	（数组）需要检查的 DOM 元素数组

返回值

由在传入的数组里不重复的元素组成的 DOM 元素数组

再次说明，jQuery 内部也使用`$.unique`函数以确保得到的元素列表不包含重复的元素。它可以在 jQuery 范围外创建的元素数组上使用。

想合并两个数组吗？没问题，我们有`$.merge`函数。

函数语法：`$.merge`

`$.merge(array1, array2)`

将第二个数组中的值合并到第一个数组中并返回结果。这个操作会修改第一个数组并将其作为结果返回

参数

`array1`	（数组）将要合并其他数组的数组
`array2`	（数组）将要被合并到第一个数组的数组

返回值

合并后的第一个数组

6

考虑：

```
var a1 = [1,2,3,4,5];
var a2 = [5,6,7,8,9];
$.merge(a1,a2);
```

这段代码执行后，`a2`没有改变，但是`a1`变成了`[1,2,3,4,5,5,6,7,8,9]`。

现在我们已经看到了 jQuery 如何简化对数组的操作，接着来看 jQuery 如何帮助我们操作普通的老式 JavaScript 对象。

6.3.6　扩展对象

虽然我们都知道 JavaScript 提供了很多特征使其在很多方面的行为类似于面向对象的语言，

但是 JavaScript 并不是人们常说的纯粹的面向对象的语言，因为它不支持某些特征。其中一个重要的特征是继承——通过扩展现有类的定义来定义新类的方法。

在 JavaScript 中模拟继承的一个模式是，通过将基础对象的属性复制到新对象来扩展对象，扩展的新对象拥有基础对象的功能。

> **注意**　如果你是“面向对象 JavaScript”的发烧友，那毫无疑问你不仅熟悉如何扩展对象的实例，而且熟悉如何通过对象构造器的 `prototype` 属性来扩展创建对象的类。`$.extend()`可以用来扩展 `prototype` 以实现基于构造器的继承，也可以用来扩展现有的对象实例从而实现基于对象的继承（jQuery 在其内部也是这样实现的）。由于理解这些高级主题不是高效使用 jQuery 的必要条件，因此虽然这个主题很重要但它却超出了本书的讨论范围。

编写执行复制操作的 JavaScript 代码来实现这种扩展相当容易，不过正如对待许多其他过程一样，jQuery 预料了到这方面的需求并提供了一个现成的实用函数来帮助解决问题：`$.extend()`。在第 7 章中我们将会看到，该函数的用途远远不止用来扩展对象。该函数的语法如下。

函数语法：`$.extend`

`$.extend(deep, target, source1, source2, ..., sourceN)`

使用其余传入的对象的属性来扩展传入的 `target` 对象

参数

`deep`	（布尔）一个可选的标志，用于决定当前执行的是深复制还是浅复制。如果省略或者为 `false`，就执行浅复制。如果为 `true`，则执行深复制
`target`	（对象）一个对象，用源对象的属性来扩展目标对象属性。在作为函数值返回之前，这个对象会直接被新属性所修改。任何与源元素中的属性具有相同名称的属性，都会被来自源元素的值所覆盖
`source1...` `sourceN`	（对象）一个或者多个对象，它或者它们的属性将会被添加到目标对象 当有一个以上的源时，在参数列表后部的源属性会覆盖在列表前部的源中具有相同名称的属性

返回值

扩展的目标对象

下面来看看这个函数的工作方式。

我们将创建 3 个对象，1 个目标和 2 个源，如下所示：

```
var target =  { a: 1, b: 2, c: 3 };
var source1 = { c: 4, d: 5, e: 6 };
var source2 = { e: 7, f: 8, g: 9 };
```

然后使用`$.extend()`操作这些对象，如下所示：

```
$.extend(target,source1,source2);
```

这个语句将会获取源对象的内容并将其合并到目标对象中。为了测试这个操作，我们在文件 chapter6/$.extend.html 中创建了示例代码，该文件会执行这段代码并在页面上显示结果。

在浏览器中加载此页面，结果如图 6-3 所示。

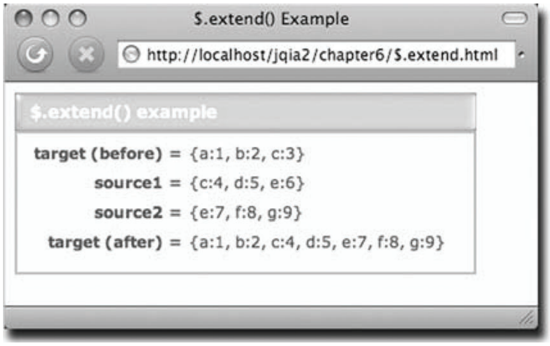


图 6-3　`$.extend()`函数不重复地合并多个源对象的属性到目标对象，并按照指定源对象的相反顺序确定对象实例的优先级

和我们看到的一样，源对象中的所有属性都被合并到 `target` 对象中。不过请注意下列的重要细微差异。

- `target` 和 `source1` 都包含一个名为 `c` 的属性。在 `source1` 中 `c` 的值替换了原始目标中的相应值。
- `source1` 和 `source2` 都包含一个名为 `e` 的属性。注意当合并到 `target` 时，`source2` 中 `e` 的值覆盖了 `source1` 中相应的值，这证明了在参数列表中后面的对象比前面的对象具有更高的优先级。

显然这个实用函数在很多场景下非常有用，在这些场景中某个对象必须用其他对象（或者一组对象）的属性进行扩展，但只有在第 7 章学习如何定义自己的实用函数时我们才会看到这个特征的一个具体和常见的用法。

在学习如何定义实用函数之前，我们还需要探讨其他的一些实用函数。

6.3.7　序列化参数值

在一个动态的、高交互性的应用中，提交请求是件屡见不鲜的事情。不过，这也是最初促使万维网发展成为网络的事情之一。

这些请求经常是作为表单提交的结果而被发送的，其中浏览器将格式化包含请求参数的请求主体。有时候，我们将<a>元素 `href` 特性的 URL 值作为请求提交。在后一种情况下，正确地创建和格式化查询字符串就成了我们的责任，这个字符串包含了我们希望在请求中包括的任何参数。

服务器端的模板工具一般都会提供强大的机制来帮助我们构建有效的 URL，但是当在客户端动态地创建 URL 时候，JavaScript 并没有提供太多的支持。请记住，我们不仅需要正确地放置组成查询字符串参数的 AND 符号（&）和等号（=），而且需要正确地对每一个名称和值进行 URI 编码。尽管 JavaScript 为此提供了一个便捷的函数（`encodeURIComponent()`），但是格式化查询字符串的任务还是落到了我们头上。

正如期望的那样，jQuery 预料到了这个困难并提供了一个工具来简化操作，即`$.param()`实用函数。

函数语法：`$.param`

`$.param(params, traditional)`

将传入的信息序列化成可在提交请求中使用的查询字符串。传入的值可以是表单元素的数组、jQuery 包装集，也可以是 JavaScript 对象。查询字符串将会被正确地进行格式化，并且字符串中的每个名称和值都会被正确地进行 URI 编码

参数

`params`	（数组| jQuery |对象）需要序列化为查询字符串的值 如果传入的是元素数组或者 jQuery 包装集，则其中包括的表单控件的名称/值对会被添加到查询字符串中。如果传入的是一个 JavaScript 对象，则对象的属性形成了参数的名称和值
`traditional`	（布尔）一个可选的标志，强制此函数按照 jQuery1.4 之前的算法来执行序列化操作。这通常只会影响有嵌套对象的源对象[①]。本节接下来会对此进行详细说明 如果省略，则默认为 `false`

返回值

格式化的查询字符串

考虑如下语句：

```
$.param({
  'a thing':'it&s=value',
  'another thing':'another value',
  'weird characters':'!@#$%^&*()_+='
});
```

在这里，我们向`$.param()`函数传递了一个包含 3 个属性的对象，其中的名称和值都包含必须进行编码的字符以确保查询字符串的有效性。这个函数的调用结果是：

```
a+thing=it%26s%3Dvalue&another+thing=another+value
➥ &weird+characters=!%40%23%24%25%5E%26*()_%2B%3D
```

① 代码 `decodeURIComponent($.param({a:{b:"c"}}, true))`的执行结果是 `a=[object+Object]`。

注意，查询字符串是如何被正确格式化的，名称和值中的非字母顺序的字符是如何被正确地编码的。对于我们而言，这可能使得字符串不具有可读性，但是服务器端却依赖于这样的字符串！

注意事项：如果传入的元素数组或者 jQuery 包装集包含非表单元素值，就将会有大量这样的条目：

```
&undefined=undefined
```

出现在结果字符串中，因为这个函数不会删除传入的参数中不合适的元素。

你可能会想这也没什么大不了的，毕竟如果这些值是表单元素，它们最终就会被浏览器通过表单来提交，那么浏览器将会处理所有的细节。好的，抓紧你的帽子[①]。在第 8 章讨论 Ajax 时，我们将看到表单元素并不总是通过它们的表单来提交！

不过这也不成问题，因为我们随后将看到 jQuery 提供了更加高级的方法[②]（该方法在内部使用这个实用函数）以便用更精细的方式来处理这类事情。

序列化嵌套参数

受到多年处理 HTTP 和 HTML 表单控件局限性的影响，Web 开发人员习惯性地认为序列化参数（也被称为查询字符串）是一个名称/值对的单层列表。

例如，设想一个收集用户名和地址的表单。对于这样的一个表单，查询参数可能包含诸如 `firstName`、`lastName` 和 `city` 等名称。查询字符串的序列化版本可能是：

```
firstName=Yogi&lastName=Bear&streetAddress=123+Anywhere+Lane
  ➥ &city=Austin&state=TX&postalCode=78701
```

这个结构序列化之前的版本可能是：

```
{
  firstName: 'Yogi',
  lastName: 'Bear',
  streetAddress: '123 Anywhere Lane',
  city: 'Austin',
  state: 'TX',
  postalCode : '78701'
}
```

作为一个对象，它并不真正代表我们所需的数据呈现方式。从数据组织的角度来看，我们可能认为这个数据由两个主要元素组成，一个是名称，另一个是地址，每一个都包含它们自己的属性。代码可能像下面这样：

```
{
  name: {
    first: 'Yogi',
    last: 'Bear'
  },
  address: {
    street: '123 Anywhere Lane',
city: 'Austin',
state: 'TX',
```

① 意思是稳住了，后面介绍的内容会彻底粉碎之前你的设想。

② 这个方法指的是 `serialize()`方法，用来从一组表单元素返回名称/值对组成的查询字符串。

```
    postalCode : '78701'
  }
}
```

但是元素的这个嵌套版本，虽然比单层的版本更加符合逻辑结构，但却不容易转化为查询字符串。

或者也很容易？

通过使用约定俗成的方括号表示法，这样的结构可以用如下形式来表示：

```
name[first]=Yogi&name[last]=Bear&address[street]=123+Anywhere+Lane
➥ &address[city]=Austin&address[state]=TX&address[postalCode]=78701
```

在这个表示法中，子属性使用方括号来表示以保持与结构的联系。很多服务器端框架如 RoR（Ruby on Rails）和 PHP，可以很方便地解码这些字符串。Java 没有原生的功能来从这种表示法中重构嵌套对象，但是创建这样的处理器很容易。

这是 jQuery1.4 的一个新行为——当向老版本 jQuery 的`$.param()`函数传递嵌套结构时，不会产生任何有意义的结果。如果希望`$.param()`使用老版本的行为，就应该设置 `traditional` 参数为 `true`。

可以在文件 chapter6/lab.$.param.html 提供的`$.param()`实验室页面中自行验证上述行为，如图 6-4 所示。

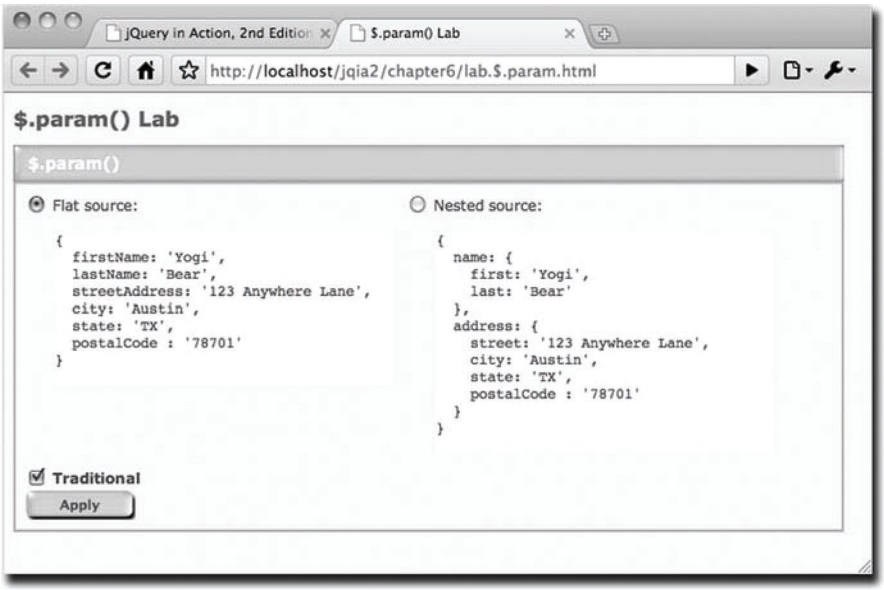


图 6-4　可以在`$.param()`实验室观察如何使用新的和传统的算法来序列化单层的和嵌套的对象

这个实验室允许我们观察`$.param()`是如何使用新算法以及传统算法来序列化单层的和嵌套的对象。

继续在这个实验室页面上练习，直到你熟悉这个函数的行为为止。我们强烈建议你复制一份这个页面来练习各种你希望序列化的对象结构。

6.3.8 测试对象

你可能已经注意到很多 jQuery 包装器方法和实用函数都有可伸缩的参数列表，可以忽略可选的参数，无需使用 `null` 值作为占位符。

以 `bind()`包装器方法为例，它的函数签名是：

```
bind(event,data,handler)
```

但是如果没有传入事件的数据，我们可以简单地将处理器函数作为第二个参数来调用 `bind()`。jQuery 通过测试参数的类型来处理这种情况，如果它发现只有两个参数，并且第二个参数是个函数，就会把第二个参数解释为处理器而不是数据参数。

如果希望创建同样友好和通用的函数和方法，那么测试参数的各种类型（包括它们是否为函数）就肯定会派上用场，因此 jQuery 公开了一些用于测试的实用函数，如表 6-4 所列。

表6-4 jQuery为测试对象提供的实用函数

函 数	描 述
`$.isArray(o)`	如果o是JavaScript数组，则返回`true`（如果o是任意其他类似数组的对象例如`jQuery`包装集，则返回`false`）
`$.isEmptyObject(o)`	如果o是不包含属性的JavaScript对象，则返回`true`，这里指的属性包括任何从`prototype`继承下来的属性[①]
`$.isFunction(o)`	如果o是JavaScript函数，则返回`true`。警告：在IE浏览器中，内置的函数例如`alert()`和`confirm()`以及元素方法都不能被正确报告为函数
`$.isPlainObject(o)`	如果o是一个通过`{}`或者`new Object()`创建的JavaScript对象，则返回`true`
`$.isXMLDoc(node)`	如果node是XML文档，或者是XML文档里的节点，则返回`true`

下面来看一些不能归入任何分类的实用函数。

6.4 其他实用函数

本节将探索一组实用函数，其中的每一个都可以定义其自己的分类。先从一个看起来好像什么都不做的函数开始。

6.4.1 什么都不做

jQuery 定义了一个什么都不做的实用函数。本可以将这个函数命名为`$.wastingAway-AgainInMargaritaville()`，但是这个名称有点长所以才命名为`$.noop()`。语法如下所示。

① 例如 `function F () {} F.prototype.name = "test";` 则`$.isEmptyObject(new F())`返回 `false`。

函数语法：$.noop

$.noop()
什么都不做
参数
无
返回值
无

唔，一个不需要参数、什么都不做并且什么都不返回的函数。有什么作用呢？

还记得有多少个 jQuery 方法需要传入可选的回调函数作为参数或者选项值吗？`$.noop()`就是在用户没有提供回调函数时作为其默认值用的。

6.4.2　测试包含关系

当希望测试一个元素是否包含在另一个元素内部时，可以使用 jQuery 提供的`$.contains()`实用函数。

函数语法：$.contains

$.contains(container, containee)
测试一个元素是否在 DOM 层次结构中包含在另一个元素内部
参数

`container`	（元素）要测试的包含另一个元素的 DOM 元素
`containee`	（元素）要测试的被包含的 DOM 元素

返回值
如果 `containee` 包含在 `container` 内部就返回 `true`，否则返回 `false`

嘿，等一下！这听起来是不是很熟悉？的确是这样，我们曾在第 2 章讨论过 `has()`方法与这个函数有惊人的相似之处。

当我们已经拿到要测试的 DOM 元素的引用并且不需要创建包装集时，这个经常在 jQuery 内部使用函数会非常有用。

下面来看看另一个和其相应的包装器方法非常相似的函数。

6.4.3　附加数据到元素上

回到第 3 章，我们曾考察了 `data()`方法，它允许我们将数据赋值给 DOM 元素。对于已经获取了 DOM 元素引用的情况，可以使用底层的实用函数`$.data()`来执行相同的操作。

函数语法：`$.data`

`$.data(element, name, value)`

使用指定的名称在传入的元素上存储或者检索数据

参数

`element`	（元素）用于存储数据的或者从中检索数据的 DOM 元素
`name`	（字符串）与数据相关联的名称
`value`	（对象）将要被赋值给指定名称的元素的数据。如果省略，则获取指定名称的数据

返回值

存储或者获取的数据值

正如所料，也可以通过一个实用函数来删除数据。

函数语法：`$.removeData`

`$.removeData(element, name)`

删除存储在传入的元素上的数据

参数

`element`	（元素）将要从中删除数据的 DOM 元素
`name`	（字符串）将要删除的数据项的名称。如果省略，则将删除所有存储的数据

返回值

无

现在把注意力转移到一个更加深奥的实用函数——显著影响事件监听器调用方式的函数。

6.4.4　预绑定函数上下文

正如在研究 jQuery 的过程中看到的，函数和其上下文在使用了 jQuery 的代码中扮演着重要角色。在随后关于 Ajax（第 8 章）以及 jQuery UI（第 9 章到第 11 章）的几章中，我们将会更加深入探讨函数，尤其是在把它们当作回调函数使用的时候。

函数的上下文（`this` 指向的对象）取决于函数是如何被调用的（如果想复习这个概念，请参见附录）。当我们想调用特定的函数并且显式地控制函数的上下文时，可以使用 `Function.call()`方法来调用这个函数。

但是如果我们不是函数的调用者，该怎么办呢？例如，如果这个函数是一个回调函数，该如何处理？在这种情况下，我们不是函数的调用者，因此不能使用 `Function.call()`来影响函数的上下文设置。

我们可以通过 jQuery 提供的一个实用函数为函数预绑定对象，这样一来当调用函数时，绑定的对象就成为了函数的上下文。这个实用函数名为`$.proxy()`，语法如下所示。

函数语法：$.proxy

$.proxy(function, proxy)

$.proxy(proxy, property)

使用预绑定的代理对象创建函数的一个副本，在函数作为回调函数被调用时，此对象作为函数的上下文

参数

function	（函数）将要使用代理对象来预绑定的函数
proxy	（对象）将要绑定为代理函数上下文的对象
property	（字符串）传入的 proxy 对象的属性名称，包含将要绑定的函数

返回值

使用代理对象来预绑定的新函数

打开位于文件 chapter6/$.proxy.html 中的示例。你将会看到如图 6-5 所示的结果。

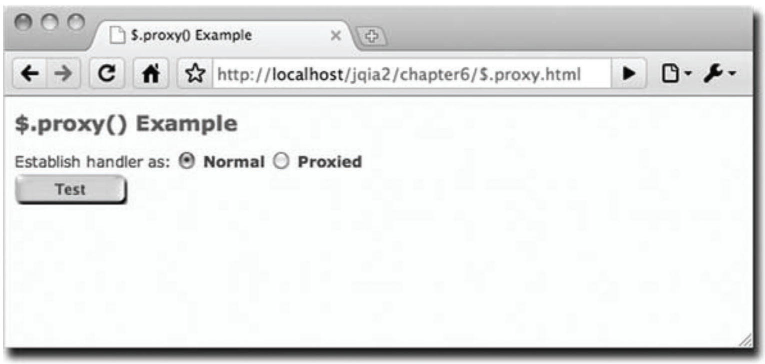


图 6-5　$.proxy 示例页面帮助我们观察普通的回调函数和代理的回调函数之间的差异

在这个示例页面中，Test 按钮被创建于 id 值为 buttonContainer 的<div>元素内。当单击 Normal 单选按钮时，在该按钮和它的容器上创建单击处理器，如下所示：

```
$('#testButton,#buttonContainer').click(
  function(){ say(this.id); }
);
```

当单击按钮时，我们希望在该按钮上调用创建的处理器，并且由于事件冒泡，也将在该按钮的父容器上调用处理器。在所有情况下，调用函数的上下文都应该是在其之上创建处理器的元素。

在处理器（用来输出函数上下文的 id 属性）中调用 say(this.id)的结果显示的一切都和预期的一样——参见图 6-6 的上半部分。处理器被调用了两次：第一次是在按钮上，第二次是在容器上，同时每个元素被分别设置为函数的上下文。

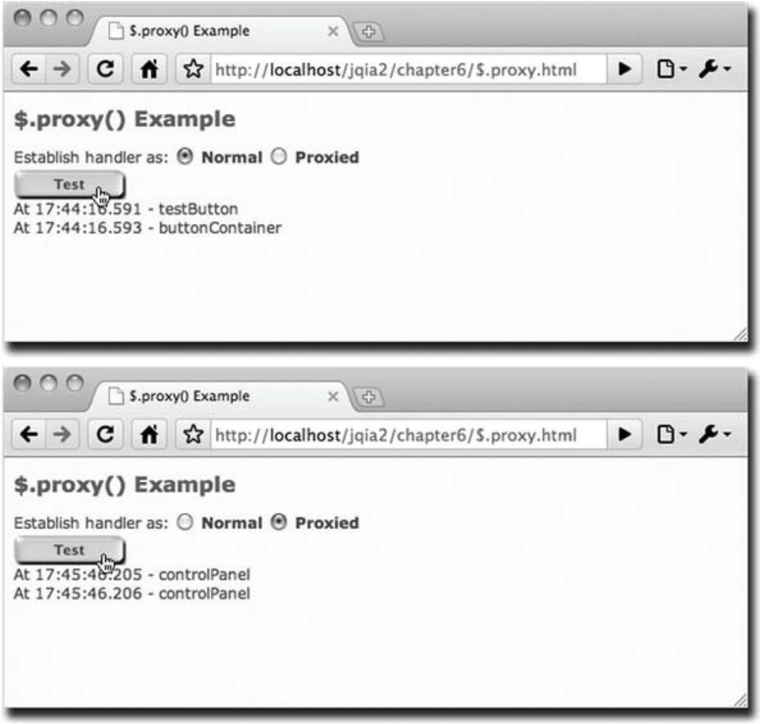


图 6-6 这个示例显示了为 Test 按钮的单击处理器预绑定对象的效果

然而，当 Proxied 单选按钮被选中时，创建处理器的方式如下：

```
$('#testButton,#buttonContainer').click(
  $.proxy(function(){ say(this.id); }, $('#controlPanel')[0])
);
```

这创建了与之前相同的处理器，不同之处在于将处理器函数传入`$.proxy()`实用函数中，同时预绑定了一个对象到该处理器。

在这个例子中，我们绑定了 `id` 为 `controlPanel` 的元素。绑定的对象也可以不是元素——事实上大部分情况下它不是元素。之所以在本例中这么做，是因为可以很容易地通过 `id` 值来找到该对象。

现在单击 Test 按钮，所看到的结果如图 6-6 下半部分所示，函数上下文已经被强制设置为通过`$.proxy()`绑定到处理器的对象。

当向回调函数提供的数据难以正常地通过闭包或者其他方式来访问时，这个功能非常有用。

`$.proxy()`的一个最常见的用途是将对象的方法绑定为处理器，并且将拥有方法的对象作为处理器的函数上下文，就如同我们直接调用该方法一样。考虑如下的对象：

```
var o = {
  id: 'o',
  hello: function() { alert("Hi there! I'm " + this.id); }
};
```

如果通过 o.hello()来调用 hello()方法，那么函数上下文（this）将是 o。但是如果将此函数作为处理器，如下所示：

```
$(whatever).click(o.hello);
```

就会发现函数的上下文是当前冒泡的元素，而不是 o。如果处理器依赖于 o，那就惨了。

可以使用$.proxy()来强制设置函数上下文为 o，以下两个语句都可以实现：

```
$(whatever).click($.proxy(o.hello,o));
```

或者

```
$(whatever).click($.proxy(o,'hello'));
```

注意，这么做意味着你将无法知道事件传播过程中当前的冒泡元素[①]——这个值通常被设置为函数的上下文。

6.4.5　解析JSON

JSON 已经迅速地成为互联网的宠儿，似乎要将看似笨拙的 XML 推下数据交换的宝座。由于大部分的 JSON 也是有效的 JavaScript 表达式语法，于是 JavaScript 的 eval()函数早就被用来将 JSON 字符串转换为等价的 JavaScript 对象。

现代浏览器提供了 JSON.parse()来解析 JSON，但是并非每个开发者都能奢望他们的所有用户都在使用最新的浏览器。jQuery 深知这一点，为此提供了$.parseJSON()实用函数。

函数语法：$.parseJSON

$.parseJSON(json)

解析传入的 JSON 字符串，返回其计算值

参数

json　　（字符串）将要解析的 JSON 字符串

返回值

JSON 字符串的计算值

如果浏览器支持 JSON.parse()，jQuery 就会使用此函数。否则，jQuery 会使用一个 JavaScript 技巧来进行求值。

记住，JSON 字符串必须是完全良好格式的，并且良好格式的 JSON 规则比 JavaScript 表达式语法的规则更加严格。例如所有的属性名称必须由双引号字符来分隔，即使它们是有效的标识符。而且必须是双引号字符——单引号字符不能用来分隔属性。无效的 JSON 将导致抛出错误。请参阅 http://www.json.org/来了解有关良好格式化 JSON 的详细信息。

说到求值……

① 其实还是有办法的，设置 hello()的第一个参数为 event，则 event.currentTarget 就是当前冒泡的元素。

6.4.6 表达式求值

虽然使用eval()会被一些互联网达人嘲笑，但是有时候它却非常管用。

eval()是在当前上下文中执行的。当编写插件或者其他可重用的脚本时，我们可能希望确保求值的过程总是在全局上下文中进行。这时候可以借助于$.globalEval()实用函数。

函数语法：$.globalEval

$.globalEval(code)

在全局上下文中对传入的 JavaScript 代码进行求值

参数

code （字符串）将要进行求值的 JavaScript 代码

返回值

JavaScript 代码的计算值

下面以一个可以为页面动态加载脚本的函数来结束对实用函数的考察。

6.4.7 动态加载脚本

大多数时候，当页面加载时我们通过页面<head>中的<script>标签来从脚本文件中加载页面所需的外部脚本。但有时候，我们可能希望事后在脚本控制下动态加载脚本。

我们可能会这么做，因为只有在特定的用户活动发生时，我们才知道需要某个脚本，并且除非绝对必要也不想包含那个脚本。也可能因为需要使用页面加载时尚不可用的信息来在不同的脚本之间进行有条件的选择。

无论是出于什么原因要在页面上动态加载脚本，jQuery 都提供了$.getScript()实用函数来简化操作。

函数语法：$.getScript

$.getScript(url, callback)

通过向指定的服务器发起 GET 请求来获取由 url 参数指定的脚本，在请求成功后（可选地）调用回调函数

参数

url （字符串）需要获取的脚本文件的 URL。URL 不局限于和容器页面相同的域名的地址

callback （函数）一个可选的函数，在脚本资源加载并且求值完毕后调用，参数为：从资源中加载的文本信息，以及一个文本的状态消息：如果所有过程都顺利进行，则为“success”

返回值

用来获取脚本的 XMLHttpRequest（XML HTTP 请求）实例

在其内部，这个函数使用 jQuery 内置的 Ajax 机制来获取脚本文件[①]。第 8 章详细阐述这些 Ajax 功能，但是不需要任何 Ajax 的知识就能使用这个函数。

获取文件后，将会对文件中的脚本进行求值，执行所有内联脚本，并且任何已定义的变量或者函数都会变得可用。

警告　在 Safari2 以及更早的版本中，从获取的文件中加载的脚本定义不会立即生效，甚至在回调函数中也是如此。任何动态加载的脚本元素不会立即生效，直到加载它的脚本块将控制权交回给浏览器。如果你的网页要支持这些早期版本的 Safari，就要作相应的调整[②]！

下面来实践一下。考虑如下脚本文件（可从 chapter6/new.stuff.js 获取）：

```
alert("I'm inline!");
var someVariable = 'Value of someVariable';
function someFunction(value) {
  alert(value);
};
```

这个小的脚本文件包含内联语句（用来弹出一个警告对话框，表明语句何时执行）、变量声明以及函数声明，当执行此函数时会弹出包含所有传入的值的警告对话框。现在来写一个页面动态地包含这个脚本文件。该页面的代码如代码清单 6-2 所示，可以在文件 chapter6/$.getScript.html 中找到。

代码清单 6-2　动态加载脚本文件并检查结果

```
<!DOCTYPE html>
<html>
  <head>
    <title>$.getScript() Example</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css" />
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
        <script type="text/javascript"
                src="../scripts/jqia2.support.js"></script>
        <script type="text/javascript">
          $(function(){
            $('#loadButton').click(function(){        ❶ 单击 Load 按钮时获取脚本
              $.getScript(
                'new.stuff.js'
                //,function(){$('#inspectButton').click()}
              );
            });
            $('#inspectButton').click(function(){     ❷ 单击 Inspect 按钮时显示结果
              someFunction(someVariable);
            });
          });
```

① 跨域的脚本文件不是通过 Ajax 获取的，而是通过向<head>添加<script>标签来加载的，详见译者博客 http://www.cnblogs.com/sanshi/archive/2011/03/02/1969224.html。

② 比如通过 setTimeout 来延迟执行后续的脚本。

```
  </script>
</head>
                                    ❸ 包含测试按钮
<body>
  <button type="button" id="loadButton">Load</button>
  <button type="button" id="inspectButton">Inspect</button>
</body>
</html>
```

这个页面定义了两个用来触发示例中的活动的按钮❸。第一个按钮的标签是 Load，该按钮会导致使用`$.getScript()`函数来动态地加载 new.stuff.js 文件❶。注意，起初第二个参数（回调函数）是被注释掉的——我们马上就会探讨这方面的内容。

单击 Load 按钮时会加载 new.stuff.js 文件，并对其内容进行求值。不出所料，脚本文件中的内联语句触发了一条警告消息，如图 6-7 所示。

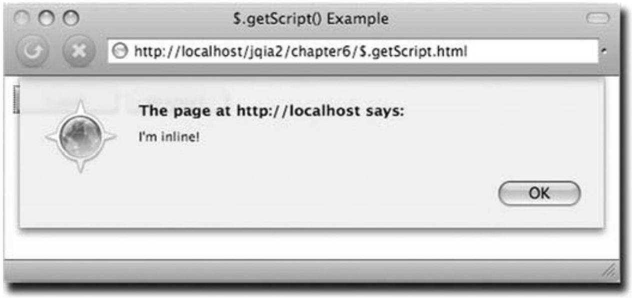


图 6-7　动态地加载并执行脚本文件，导致内联的警告对话框语句被执行

单击 Inspect 按钮后会执行其`click`处理器❷，该处理器会执行动态加载的`someFunction()`函数，并向此函数传递动态加载的`someVariable`变量的值。如果警告对话框如图 6-8 所示，我们就知道变量和函数都已经被正确加载了。

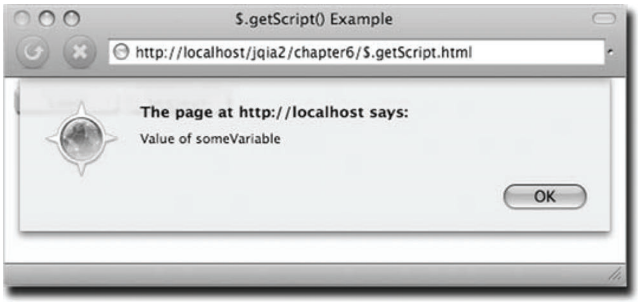


图 6-8　警告对话框的出现表明动态加载函数成功，而显示内容正确表明变量已经被动态地加载

如果你仍然运行 Safari 2 或者更早版本的浏览器（目前看来已经非常过时），并且希望观察我们之前曾经提醒过你的 Safari 早期版本的行为，那么可以复制一份图 6-8 所示页面的 HTML 文件，

并取消其中对$.getScript()函数中回调参数的注释。这个回调会执行 Inspect 按钮的 click 处理器，将加载的变量作为参数来调用动态加载的函数。

在非 Safari 2 的浏览器中，从脚本中动态加载的函数和变量在回调函数中是可用的。但是在 Safari 2 中执行时，什么也没有发生！当在 Safari 较早版本的浏览器中使用$.getScript()函数时，我们需要留意这种功能上的分歧。

6.5　小结

在本章中，我们考察了除操作由匹配 DOM 元素组成的包装集的方法外，jQuery 提供的一些特征。它们包括各种各样的函数，以及一组直接定义在 jQuery 顶级名称（以及别名$）上的标志。

我们看到 jQuery 如何通过$.support 对象上的各种标志来告诉我们当前浏览器的功能。当需要求助于浏览器检测来了解$.support 无法提供的、浏览器在功能和运行上的差异时，$.browser 提供的一组标志能够帮助我们确定当前显示页面的浏览器家族。当不可能按照独立于浏览器的方式编写代码时，并且首选的特征检测方法行不通时，才应该考虑浏览器检测这个最后的手段。

jQuery 认识到网页开发者有时可能需要同时使用 jQuery 和其他库，为此提供了$.noConflict()用来允许其他库使用$别名。在调用此函数后，所有的 jQuery 操作必须使用 jQuery 名称而不能使用$。

$.trim()的存在是为了填补原生 JavaScript 的 String 类的一个空白，它用来修剪字符串开始和结尾处的空白字符。

jQuery 也提供了一组非常有用的函数用来操作数组中数据集。$.array()可以很方便地遍历数组中的每一项。$.grep()允许我们使用任何筛选条件来筛选源数组中的数据，从而创建新的数组。$.map()允许我们在源数组上轻松使用自定义的转换函数，从而产生包含转换后的值的新数组。

我们可以使用$.makeArray()将 NodeList 实例转换为 JavaScript 数组，可以使用$.inArray()来测试一个值是否在数组中，甚至可以通过$.isArray()来测试某个值是否是数组，也可以使用$.isFunction()来测试一个值是否为函数。

我们也看到了 jQuery 是如何通过$.param()来创建正确格式化和编码的查询字符串。

jQuery 提供了$.extend()函数用来合并对象，甚至可以用来模拟某种继承架构。这个函数可以将多个源对象的属性合并到目标对象中。

我们也看到了很多用来测试对象是否为函数、JavaScript 对象，甚至是否为空对象的函数——这在许多情况下都很有用，特别是在检查变量参数列表的时候。

$.proxy()方法用来预绑定一个对象，随后用于事件处理器调用的函数上下文，$.noop()函数什么也不做。

当我们需要动态加载脚本文件的时候，可以使用 jQuery 定义的$.getScript()来在页面生命周期的任何时候加载和执行脚本文件，甚至是在页面所在域之外其他域的脚本。

有了这些附加工具在手，我们就能为 jQuery 添加自定义扩展。我们将在下一章中学习这方面的内容。

第 7 章

扩展 jQuery

本章内容

- 为什么要使用自定义代码来扩展 jQuery
- 有效扩展 jQuery 的准则
- 编写自定义的实用函数
- 编写自定义的包装器方法

在前面几章中，我们看到 jQuery 提供了一个包含了有用方法和函数的庞大工具箱，我们可以很容易地组合使用这些工具来为页面添加任何选定的行为。有时我们需要代码遵循重复使用的常用模式。当这种模式出现时，捕捉这些重复的操作来创建可重用的工具是有意义的，可以将其添加到原始的工具箱中。本章中，我们将探讨如何捕捉这些可重用的代码片段并实现 jQuery 扩展。

不过在此之前，首先来讨论下为什么要将代码模式化以便将其实现为 jQuery 扩展。

7.1　为什么要扩展 jQuery

如果在通读本书时留心观察，并且已经审查了其中的代码示例，那你肯定会注意到在页面中采用 jQuery 对编写脚本的方式有着深刻的影响。

jQuery 提倡一种编写页面代码的风格：通常是形成元素的包装集然后应用 jQuery 方法，或者为该集合应用方法链。当编写代码时，我们可以根据个人喜好随意而为，但有经验的开发者都赞同，使网站上的所有代码或者至少是绝大多数代码保持一致的风格是良好的做法。

因此将代码模式化来实现为 jQuery 扩展的一个很好的理由是，为了帮助整个网站保持一致的代码风格。

理由不够充分？还需要更多的理由？jQuery 的全部要点是提供一套可重用的工具和 API。jQuery 的创建者精心策划了库的设计和如何布置工具来提升可重用性的理念。通过遵循这些工具的设计先例，自然就能从这些设计中获得规划的好处——这是将代码实现为 jQuery 扩展的有说服力的第二个理由。

还是不能令人信服？我们考虑的最后一个理由（虽然可能其他人可以列出更多的理由）是，通过扩展 jQuery，就可以利用 jQuery 提供的现有代码基础。例如，通过创建新的 jQuery 方法（包装器方法），我们自动继承了 jQuery 强大的选择器机制。当可以依靠 jQuery 已经提供的强大工具

的时候为什么还要从头开始编写所有代码呢？

综上所述，将可重用的组件实现为 jQuery 扩展是一个很好的做法和聪明的工作方式。在本章的其余部分，我们将研究允许创建 jQuery 插件的准则和模式，并会创建一些 jQuery 插件。第 8 章会包含一个完全不同的主题（Ajax），我们将看到更多的证据，可以证明在现实世界中把可重用的组件实现为 jQuery 插件有助于保持代码的一致性，并且使得从一开始编写这些组件变得更加容易。

不过先来看下开发指南……

7.2　jQuery 插件开发指南

标志！标志！到处都是标志！

阻隔了风景，破坏了心情。

这么做！不要那么做！难道你看不懂标志吗？

——五人电子乐队（Five Man Electric Band），1971

尽管五人电子乐队在 1971 年狂热地宣称了反对正统流派的立场，但是有时规则是一件好事。不以规矩，不成方圆。

因此 jQuery 插件开发需要有规则（更像是常识性指南），用来管理如何使用自定义插件来扩展 jQuery。这些规则不仅帮助确保将新代码正确地嵌入到 jQuery 架构，而且确保新代码与其他 jQuery 插件甚至其他 JavaScript 库能够在一起正常工作。

扩展 jQuery 有两种形式：

- 直接定义在`$`（`jQuery` 的别名）上的实用函数；
- 操作 jQuery 包装集的方法（也就是 jQuery 方法）。

在本节的其余部分，我们将检查一些适用于这两种类型扩展的公用准则。然后在后面几节中，我们将探讨特定于编写每一种类型插件的准则和技术。

7.2.1　为文件和函数命名

To Tell the Truth（实话实说）是一档起始于 20 世纪 50 年代的美国电视游戏节目，其中许多参赛者都自称是姓名相同的同一个人，一组名人负责确定参赛者中哪一个确实是现实中大家都声称的那个人。尽管对于电视观众而言这很有意思，但是在编程中名称冲突却一点也不好玩。

我们将讨论在插件内部避免名称冲突的方法，但是首先来给将要编写的插件起个文件名以便不会和其他文件冲突。

这个由 jQuery 团队推荐的指南不仅简单，而且高效。它提倡使用如下格式：

- 为文件名添加 jquery 前缀；
- 前缀后面是插件的名称；
- 包含插件的主版本号和次版本号；（可选项）
- 以.js 结束。

例如，如果希望编写一个以 Fred 命名的插件，这个插件的 JavaScript 文件名可以是 jquery.fred-1.0.js。jquery 前缀的使用消除了与其他库使用的文件之间可能产生的名称冲突。毕竟，任何人写的非 jQuery 插件都没有理由使用 jquery 作为前缀，不过在 jQuery 社区内部这样的命名还是有可能会产生冲突。

当编写自己使用的插件时，需要做的全部事情就是避免和计划使用的任何其他插件产生冲突。但是当计划编写公开给其他人使用的插件时，我们需要避免和任何已经发布的插件产生冲突。

避免冲突的最好方式是与 jQuery 社区内现有插件保持协调。一个良好的开端是从 http://plugins.jquery.com/页面开始，不过除了了解社区已有插件外，还有其他可以采取的预防措施。

确保插件名称不与其他插件名称发生冲突的一个方法，是使用对于我们或者组织而言唯一的名称来为插件名称添加子前缀。例如，本书中开发的所有插件都使用文件名前缀 jquery.jqia（jqia 是 jQuery in Action 的首字母缩写），确保它们不会和任何其他插件名称冲突，任何人都可以在自己的应用程序中使用这些插件。同样，jQuery 表单插件的文件以 jquery.form 前缀开头。并非所有的插件都遵循这个约定，但是随着插件数目的增加，遵循这种约定将变得越来越重要。

不管它们是新的实用函数还是操作 jQuery 包装器上的方法，我们给函数取名时也需要采取类似的措施。

当创建自己使用的插件时，我们通常知道将要使用的其他插件；因此避免名称冲突是一件容易的事情。但是如果创建用于公共用途的插件该怎么办？或者如果我们最初打算创建私有用途的插件，结果证明它们非常有用以至于我们想和社区的其他用户分享这些插件该怎么办？

再次强调，熟悉已经存在的插件对于避免 API 冲突大有帮助，但是我们也鼓励将相关函数的集合收集到一个公共的前缀之下（类似于对文件名的建议），以避免命名空间混乱。

现在，该如何处理$冲突？

7

7.2.2　当心$

“真正的$站起来好吗？”

写了相当多的 jQuery 代码，我们已经看到使用$别名来代替 jQuery 有多么方便。但是当编写可能会用作其他人页面中的插件时，我们就不能如此的漫不经心了。作为插件作者，我们无法知道 Web 开发者是否打算使用$.noConflict()函数来允许$别名被另一个库使用。

可以采用一刀切的做法来使用 jQuery 名称代替$别名，但可恶的是，我们喜欢使用$，不愿意将它拱手相让。

第 6 章引入了一个习惯用法，它经常被用来确保在局部区域内让$别名指向 jQuery 名称，而不会影响页面的其余部分，这个小技巧也可以（经常）用来定义 jQuery 插件，如下所示：

```
(function($){
//
// 插件定义
//
})(jQuery);
```

通过向参数为$的函数传递 jQuery，从而确保在函数主体内$是对 jQuery 的引用。

现在，我们可以高高兴兴地在插件定义中使用$了。

在深入学习如何为 jQuery 添加新的元素之前，先来看下我们鼓励插件作者使用的另一项技术。

7.2.3　简化复杂参数列表

大部分的插件喜欢简洁，要么没有参数，要么只有几个参数。我们已经在大部分的核心 jQuery 方法和函数中看到充分的证据，它们要么只需要少数几个参数要么一个参数都不需要。智能的默认值会在可选的参数被省略时提供，并且当一些可选的参数被省略时，参数的顺序甚至可能代表不同的含义。

bind()方法就是一个很好的例子。如果省略可选的数据参数时，通常被指定为第三个参数的监听器函数可以作为第二个参数来提供。JavaScript 动态性和解释性允许我们编写这样灵活的代码，但是当参数的数目变大时这种事情开始变得容易出错和复杂（对于 Web 开发者和插件作者同样如此）。当很多参数都是可选的时候出错的可能性也随之增加。

考虑如下有点复杂的函数签名：

```
function complex(p1,p2,p3,p4,p5,p6,p7) {
```

这个函数接受 7 个参数，假设除了第一个其他参数都是可选的。当省略可选的参数时，就会存在过多的可选参数，这样就不能根据调用者的意图做出任何智能的猜测。如果该函数的调用者只是省略了结尾的参数，那就不成问题，因为尾部的可选参数可以被检测为 null。但是如果调用者想指定 p7，而让 p2 到 p6 保持默认值该怎么办？调用者需要为省略的参数使用占位符，如下所示：

```
complex(valueA,null,null,null,null,null,valueB);
```

哎呀！更糟糕的调用是这样的：

```
complex(valueA,null,valueC,valueD,null,null,valueB);
```

以及由这种性质决定的其他变体。使用这个函数的 Web 开发者不得不小心翼翼地跟踪计算 null 的个数和参数的顺序，另外也很难阅读和理解这个代码。

但是除了不给调用者提供过多的选项以外，还有什么办法呢？

JavaScript 灵活性再次派上用场；平息这种混乱的模式已在页面开发社区兴起——选项散列值。使用这种模式，就可以将可选的参数收集到以 JavaScript 的 Object 实例来呈现的单个参数中，Object 实例的名称/值对属性作为可选的参数。

使用这种技术，第一个示例可以这么写：

```
complex(valueA, {p7: valueB});
```

第二个可以这么写：

```
complex(valueA, {
  p3: valueC,
  p4: valueD,
  p7: valueB
});
```

这好多了！

我们不用必须为省略的参数使用占位符 null，也不再需要计算参数的个数；每个可选的参数都被贴上了标签以便清晰地表明该参数代表了什么（当使用比 p1 ~ p7 更好的参数名称的时候）。

注意 一些 API 遵循这个约定——将可选参数打包到单个的 options 参数（留下必需的参数作为单独的参数），而其他 API 将完整的参数集合（必需的和可选的做同样处理）打包到单个对象中。这两种方式各有各的长处，因此请选择一个最适合你代码的方式。

尽管对于复杂函数的调用者而言这明显有很大的优势，但是这对于插件作者会有什么后果呢？事实证明，jQuery 提供的机制可以用来将这些可选参数收集在一起并将其与默认值合并。下面重新考虑拥有 1 个必需参数和 6 个可选参数的复杂的示例函数。简化后的新函数签名如下所示：

```
complex(p1,options)
```

在函数中，我们可以使用便捷的$.extend()实用函数将这些选项与默认值合并。考虑如下代码：

```
function complex(p1,options) {
  var settings = $.extend({
    option1: defaultValue1,
    option2: defaultValue2,
    option3: defaultValue3,
    option4: defaultValue4,
    option5: defaultValue5,
    option6: defaultValue6
  },options||{});
  // 函数的剩余部分……
}
```

通过把 Web 开发者传递到 options 参数的值与包含所有可用选项默认值的对象合并，settings 变量最终成为由 Web 开发者指定的任何显式值所取代的默认值。

提示 也可以只使用 options 引用自身来存储这个值，而不是创建一个新的 settings 变量。那样就可以减少一个在栈上的引用，这里为了保持代码的清晰，先不采用这种方式。

注意，我们使用||{}来防止 options 对象是 null 或 undefined，如果 options 求值为 false（众所周知 null 和 undefined 正是如此）则提供一个空对象。

简单、通用并且对调用者友好！

在本章的稍后部分和第 8 章将要介绍的 jQuery 函数中，我们将看到使用这个模式的示例，但是目前先来看看如何使用自定义实用函数来扩展 jQuery。

7.3 编写自定义实用函数

本书中，我们使用术语*实用函数*来描述定义为 jQuery（因此也是$）属性的函数。这些函数

不是为了操作 DOM 元素（那是为操作 jQuery 包装集而定义的方法的工作），它们或者是操作非元素的 JavaScript 对象，或者是执行一些其他的不具体操作任何对象的行为。我们看到过的这些类型的函数分别有`$.each()`和`$.noConflict()`。

本节中，我们将学习如何添加类似的自定义函数。

将函数添加为 `Object` 实例的属性很简单,就如同声明函数并把它赋值给 `Object` 属性一样。（如果这看起来像黑魔法，并且你还没有通读附录，现在正是通读附录的好时机。）创建普通的自定义实用函数非常简单：

```
$.say = function(what) { alert('I say '+what); };
```

事实上，它就是那么的容易。但是这种定义实用函数的方式不是没有缺陷的。记得 7.2.2 节中关于`$`的讨论吗？如果开发者在使用 Prototype 的页面包含这个函数并且已经调用过`$.noConflict()`该怎么办？我们将会在 Prototype 的`$()`函数上创建一个方法，而不是添加了一个 jQuery 扩展。（如果概念“函数的方法”让你头疼的话，请通读附录。）

对于永远不会公开的私有函数来说，这没有问题。但是即使那样，如果将来对页面的改变导致重新分配`$`该怎么办？小心谨慎不出错才好。

确保其他人对`$`的改变不会影响到我们的一种方式是完全避免使用`$`。可以像下面这样编写那个普通的函数：

```
jQuery.say = function(what) { alert('I say '+what); };
```

这看起来似乎是一种简单的解决办法，但事实证明对于复杂的函数这决不是最佳的解决方案。如果函数主体内部使用了大量 jQuery 方法和函数来完成工作，该怎么办？我们需要在函数内部自始至终使用 `jQuery` 而不是`$`。这相当啰嗦且不优雅，此外一旦使用过`$`，就会对它爱不释手。

回顾 7.2.2 节介绍的习惯用法，可以按照如下方式安全地编写函数：

```
(function($){
  $.say = function(what) { alert('I say '+what); };
})(jQuery);
```

我们强烈推荐使用这种模式（尽管对于如此简单的一个函数而言有点大材小用），因为它可以在声明和定义函数时保护`$`的使用。如果函数需要变得更加复杂，我们可以扩展和修改它，而无需担心使用`$`是否安全。

我们现在趁热打铁，马上来实现一个较复杂的自定义实用函数。

7.3.1　创建数据操作的实用函数

当进行定宽输出的时候，需要格式化一个数字值以适合定宽的字段（这里的宽度定义为字符的个数）。这种操作经常会在定宽字段中向右对齐数字值，并为数字值添加足够的填充字符前缀来弥补数字值长度和字段长度之间的差异。

下面来编写这样的一个实用函数，语法如下所示。

函数语法：$.toFixedWidth

$.toFixedWidth(value, length, fill)

把传入的值格式化为指定长度的定宽字段。可以提供一个可选的填充字符。如果数字值的长度超过了指定的长度，它的高位数字将被截断以便符合指定的长度

参数

value	（数字）将要格式化的值
length	（数字）结果字段的长度
fill	（字符串）对数字值进行前（左）填充时使用的填充字符。如果省略，则默认为 0

返回值

定宽字段

这个函数的实现如代码清单 7-1 所示。

代码清单 7-1 实现$.toFixedWidth()自定义实用函数

```
(function($){
 $.toFixedWidth = function(value,length,fill) {
   var result = (value || '').toString();
   fill = fill || '0';                              ❶ 分配默认值
   var padding = length - result.length;          ❷ 计算填充长度
   if (padding < 0) {
     result = result.substr(-padding);           ❸ 如果必要则截断字符串
   }
   else {
     for (var n = 0; n < padding; n++)
       result = fill + result;                    ❹ 填充结果
   }
   return result;              ❺ 返回最终结果
 };
})(jQuery);
```

这个函数简单明了。传入的值被转换为它的等价字符串，填充字符由传入的值或默认值 0 共同决定❶。然后计算需要填充的字符个数❷。

如果最终拿到的填充数是负数（结果值比传入字段的长度还要长），则从结果值的开头截断字符串最终得到指定长度的字符串❸；否则在把结果值作为函数结果返回之前❺，在其开头补足适当个数的填充字符❹。

为实用函数添加命名空间

如果想确保自己的实用函数不会和其他人的实用函数冲突，可以通过在$上创建命名空间对象来为函数添加命名空间，反过来说这个命名空间充当了函数的拥有者。例如，如果我们想将所有的日期格式化函数放在名为 jQiaDateFormater 的命名空间中，可以这么做：

```
$.jQiaDateFormatter = {};
$.jQiaDateFormatter.toFixedWidth = function(value,length,fill) {...};
```

这确保了像 toFixedWidth()这样的函数永远不会和另一个名字类似的函数冲突。（当然了，我们仍然需要为命名空间冲突担忧，但是这个很容易处理。）

7

上面这个简单的示例向我们展示了添加实用函数是多么地容易。但还是会有改进的余地。考虑如下练习题。

(1) 像本书中的大部分示例那样，错误检测被减到最小以便关注眼前的示例。你将如何改进这个函数以便应对调用者的错误（例如没有为 `value` 和 `length` 传入数字类型的值）呢？如果调用者完全不传递这些参数，该怎么办？

(2) 我们很小心地截断过长的数字值，以便确保结果总是符合指定的长度。但是如果调用者为填充字符传递多于单个字符的字符串，那就前功尽弃了。你该如何处理这种情况？

(3) 如果不想截断过长的数字值该怎么办？

现在来处理一个更加复杂的函数，我们可以在其中使用刚刚编写的`$.toFixedWidth()`函数。

7.3.2　编写日期格式器

如果由服务器端转向客户端编程世界，那么你盼望已久的一件事情可能是方便的日期格式器；JavaScript 的 `Date` 类型不提供的这个功能。因为这样的一个函数将要操作 `Date` 实例而不是任何的 DOM 元素，因此它是实用函数的完美候选者。下面编写这样的函数，语法如下所示。

函数语法：`$.formatDate`

`$.formatDate (date, pattern)`

根据提供的模式格式化传入的日期。在模式中被替换的标记如下所示。

yyyy：4 位数字的年份

yy：2 位数字的年份

MMMM：月份的完整名称

MMM：月份的缩写名称

MM：由 0 补足的、2 个字符字段的月份数字

M：月份数字

dd：由 0 补足的、2 个字符字段的月份中的天

d：月份中的天

EEEE：星期中天的完整名称

EEE：星期中天的缩写名称

a：上午或者下午（AM 或者 PM）

HH：由 0 补足的、2 个字符字段的一天中 24 小时制的第几个小时

H：一天中 24 小时制的第几个小时

hh：由 0 补足的、2 个字符字段的一天中 12 小时制的第几个小时

h：一天中 12 小时制的第几个小时

mm：由 0 补足的、2 个字符字段的一小时中的分钟数

m：一小时中的分钟数

ss：由 0 补足的、2 个字符字段的一分钟内的秒数

s：一分钟内的秒数

S：由 0 补足的、3 个字符字段的一秒内的毫秒数

参数

date	（日期）将要格式化的日期
pattern	（字符串）用来格式化日期的模式。任何不匹配模式标记的字符被原封不动地复制到结果中

返回值

格式化后的日期

这个函数的实现显示在代码清单 7-2 中。我们不打算非常详细地介绍用来执行格式化的算法（毕竟，这不是介绍算法的图书），不过我们将通过这个实现指出一些有趣的策略，在创建稍微复杂的实用函数时会用到这些策略。

代码清单 7-2 $.formatDate()自定义实用函数的实现

```
(function($){
  $.formatDate = function(date,pattern) {          ❶ 实现函数主体
    var result = [];
    while (pattern.length > 0) {
      $.formatDate.patternParts.lastIndex = 0;
      var matched = $.formatDate.patternParts.exec(pattern);
      if (matched) {
        result.push(
           $.formatDate.patternValue[matched[0]].call(this,date)
        );
        pattern = pattern.slice(matched[0].length);
      } else {
        result.push(pattern.charAt(0));
        pattern = pattern.slice(1);
      }
    }
  return result.join('');
};                                                 ❷ 定义正则表达式
$.formatDate.patternParts =
  /^(yy(yy)?|M(M(M(M)?)?)?|d(d)?|EEE(E)?|a|H(H)?|h(h)?|m(m)?|s(s)?|S)/;

$.formatDate.monthNames = [                        ❸ 提供月份的名称
  'January','February','March','April','May','June','July',
  'August','September','October','November','December'
];                                                 ❹ 提供星期中每天的名称
$.formatDate.dayNames = [
  'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday',
  'Saturday'
];                                                 ❺ 收集“标记到值”的转换函数
$.formatDate.patternValue = {
  yy: function(date) {
    return $.toFixedWidth(date.getFullYear(),2);
  },
  yyyy: function(date) {
    return date.getFullYear().toString();
  },
  MMMM: function(date) {
    return $.formatDate.monthNames[date.getMonth()];
  },
```

7

```
    MMM: function(date) {
      return $.formatDate.monthNames[date.getMonth()].substr(0,3);
    },
    MM: function(date) {
      return $.toFixedWidth(date.getMonth() + 1,2);
    },
    M: function(date) {
      return date.getMonth()+1;
    },
    dd: function(date) {
      return $.toFixedWidth(date.getDate(),2);
    },
    d: function(date) {
      return date.getDate();
    },
    EEEE: function(date) {
      return $.formatDate.dayNames[date.getDay()];
    },
    EEE: function(date) {
      return $.formatDate.dayNames[date.getDay()].substr(0,3);
    },
    HH: function(date) {
      return $.toFixedWidth(date.getHours(),2);
    },
    H: function(date) {
      return date.getHours();
    },
    hh: function(date) {
      var hours = date.getHours();
      return $.toFixedWidth(hours > 12 ? hours - 12 : hours,2);
    },
    h: function(date) {
      return date.getHours() % 12;
    },
    mm: function(date) {
      return $.toFixedWidth(date.getMinutes(),2);
    },
    m: function(date) {
      return date.getMinutes();
    },
    ss: function(date) {
      return $.toFixedWidth(date.getSeconds(),2);
    },
    s: function(date) {
      return date.getSeconds();
    },
    S: function(date) {
      return $.toFixedWidth(date.getMilliseconds(),3);
    },
    a: function(date) {
      return date.getHours() < 12 ? 'AM' : 'PM';
    }
  };
})(jQuery);
```

除了一些用于控制代码量的 JavaScript 技巧之外，这个实现最有趣的地方是需要一些辅助数据来完成工作的函数❶。特别是以下各方面：

- 用来匹配模式中标记的正则表达式❷；
- 月份的英文名称列表❸；
- 星期几的英文列表❹；
- 一组子函数来（通过给定的源日期）为每个标记类型提供值❺。

我们可以在函数主体中通过 `var` 定义来包含每个数据，但是这可能使已经很棘手的算法更加凌乱，因为它们都是常量，因此将它们从可变数据中分离开来是有意义的。

我们不想让只有这个函数会用到一些名称污染全局命名空间，甚至是$命名空间，因此在新函数①上声明这些属性。记住，JavaScript 函数是一等的对象，它们可以像任何其他 JavaScript 对象一样拥有自己的属性。

至于格式算法本身？简而言之，它的操作如下。

(1) 创建一个数组来保持结果的各个部分。

(2) 遍历模式，用掉已识别的标记和非标记字符，直到模式检查完毕。

(3) 通过设置正则表达式的 `lastIndex` 属性为 0，来在每次遍历时重置它（存储在 `$.formatDate.patternParts` 中）。

(4) 为了匹配模式当前开头处的标记，测试正则表达式。

(5) 如果发生匹配，就从`$.formatDate.patternValue`转换函数集合中调用函数，从 `Date` 实例获取适当的值。将这个值添加到结果数组的末尾，并且从模式的开头删除匹配的标记。

(6) 如果没有在模式的当前开头位置匹配到标记，则从模式中删除第一个字符并且把它添加到结果数组的末尾。

(7) 当整个模式被用光的时候，把结果数组连接成一个字符串，并作为函数的值返回。

注意，`$.formatDate.patternValue` 集合中的转换函数使用了前一节创建的`$.toFixedWidth()`函数。

可以在文件 chapter7/jquery.jqia.dateFormat.js 中找到这两个函数，chapter7/test.dateFormat.html 是用于测试这个插件的基本页面。

操作普通的 JavaScript 对象也不错，但是 jQuery 的真正力量在于操作 DOM 元素集合的包装器方法（通过强大的 jQuery 选择器创建的）。下面来看下如何添加自定义的强大的包装器方法。

7.4　添加新的包装器方法

jQuery 的真正威力在于方便快捷地选择和操作 DOM 元素的能力。幸运的是，我们可以通过添加自己的包装器方法来扩展这种能力，这些包装器方法是用来操作我们认为合适的选定的 DOM 元素。通过添加包装器方法，我们可以使用强大的 jQuery 选择器选取要在哪些元素上进行操作，而无需自己完成所有的工作。

① 这个新函数指的就是`$.formatDate`函数。

使用已有的 Javascript 知识，我们大概可以知道如何添加实用函数到$命名空间，但是对于包装器方法这不一定是对的。我们需要知道 jQuery 特有的一件事情：要给 jQuery 添加包装器方法，必须把这些方法赋值给$命名空间中名为 fn 的对象的属性。

创建包装器函数的一般模式为：

```
$.fn.wrapperFunctionName = function(params){function-body};
```

下面构建简单的包装器方法将匹配 DOM 元素的颜色设置为蓝色：

```
(function($){
  $.fn.makeItBlue = function() {
    return this.css('color','blue');
  };
})(jQuery);
```

与创建实用函数一样，我们在确保$为 jQuery 别名的外部函数内声明包装器方法。但是和实用函数不同，我们将新的包装器方法创建为$.fn 属性，而不是$的属性。

注意　如果你熟悉“面向对象的 JavaScript”及其基于原型的类声明，你可能有兴趣知道$.fn 仅仅是对象内部 prototype 属性的别名，jQuery 使用这种方式来创建其包装器对象。

在该方法的主体内，函数上下文（this）指向包装集。我们可以在该方法上使用所有预定义的 jQuery 方法。在本例中，我们在包装集上调用了 css()方法来设置所有已匹配 DOM 元素的 color 为 blue。

警告　包装器主体部分中的函数上下文（this）指向包装集，但是当在这个函数内声明内联函数时，每一个内联函数都有其自己的函数上下文。在这种情况下使用 this 时必须小心以确保它指向的是你所想的函数上下文！例如，如果使用 jQuery 方法 each()及其迭代器函数，则在迭代器函数内的 this 指向的是当前迭代的 DOM 元素。

我们可以对包装集的 DOM 元素进行几乎任何想要的操作，但是在定义新的包装器方法时还有一个非常重要的原则：除非要返回特定的值，否则函数应该总是将包装集作为其返回值。这允许新方法参与到任何 jQuery 方法链中。在示例中，因为 css()方法返回的是包装器，所以返回 css()的调用结果即可。

在之前的示例中，我们使用 this 将 jQuery 的 css()方法应用到包装集中的所有元素上。如果出于某种原因，需要单独处理每个包装元素（可能是因为要根据条件来决定处理过程），那么可以使用如下模式：

```
(function($){
  $.fn.someNewMethod = function() {
    return this.each(function(){
      //
```

```
        // 这里是函数主体，this 指向单个的 DOM 元素
        //
      });
    };
  })(jQuery);
```

在这个模式中，each()方法用来遍历包装集中每个单独的元素。注意，在迭代器函数内 this 指向当前 DOM 元素而不是整个包装集。each()返回的包装集作为新方法的值，因此这个新方法可以参与到方法链中。

下面思考一个之前提到的设置元素颜色为蓝色的示例的变体，它可以单独处理每个元素：

```
(function($){
  $.fn.makeItBlueOrRed = function() {
    return this.each(function(){
      $(this).css('color',$(this).is('[id]') ? 'blue' : 'red');
    });
  };
})(jQuery);
```

在这个变体里，我们想根据每个元素的独特条件（在这个案例中，条件是元素是否拥有 id 特性）来设置颜色，颜色可以是 blue 或 red，这需要遍历包装集以便单独检测和操作每个元素。

通过使用接受函数作为值的方法来进行遍历

注意，blue 或 red 示例用来展示如何使用 each()来遍历包装集中的单个元素有点牵强。因为 css()方法接受函数作为值（该函数会自动遍历这些元素），聪明的你可能已经注意到这个自定义方法可以不使用 each()来编写，如下所示：

```
(function($){
  $.fn.makeItBlueOrRed = function() {
    return this.css('color',function() {
      return $(this).is('[id]') ? 'blue' : 'red';
    });
  };
})(jQuery);
```

这是一个在 jQuery API 中常见的习惯用法。当传递一个函数来取代某个值的时候，会以遍历包装集中元素的形式来调用该函数。

使用 each()的示例变体展示了没有这种自动遍历元素的情况。

这就是创建包装器方法的全部内容，但是（怎么总是有但是呢？）在创建更加复杂的 jQuery 包装器方法时，还有一些技巧需要注意。下面再定义另外两个更加复杂的插件方法来考察这些技巧。

7.4.1 在包装器方法中应用多个操作

接下来开发另一个新的插件方法用来在包装集上进行多个操作。假设需要切换表单中的文本字段的只读状态，同时改变字段的外观。可以将几个现有的 jQuery 方法链接起来轻松完成这个任务，但是我们想要整齐利落地完成这个任务并将这些操作绑定到单个方法里。

我们将新方法命名为 setReadOnly()，其语法如下所示。

方法语法：setReadOnly

setReadOnly(state)

设置包装文本字段的只读状态为 state 指定的状态。调整字段的不透明度：如果不处于只读状态，则设置为 100%；如果处于只读状态，则设置为 50%。忽略包装集中除了文本字段以外的任何元素

参数

state	（布尔）要设置的只读状态。如果为 true，则设置文本字段为只读状态；否则清空只读状态

返回值

包装集

这个插件的实现如代码清单 7-3 所示，也可以在文件 chapter7/jquery.jqia.setreadonly.js 中找到。

代码清单 7-3　setReadOnly()自定义包装器方法的实现

```
(function($){
  $.fn.setReadOnly = function(readonly) {
    return this.filter('input:text')
      .attr('readOnly',readonly)
      .css('opacity', readonly ? 0.5 : 1.0)
      .end();
  };
})(jQuery);
```

这个示例只比最初的示例稍微复杂了点，但是它展示了下列关键概念。

- 传入的参数影响方法的操作。
- 通过使用 jQuery 链来对包装集应用 4 个 jQuery 方法。
- 新的方法可以参与 jQuery 链，因为该方法将包装集作为其返回值。
- filter()方法用来确保无论 Web 开发者向包装器方法应用了什么包装元素集，只有文本字段受影响。
- 调用 end()方法以便将原始的包装集（并非过滤后的包装集）作为调用的值返回。

该如何使用这个方法呢？

通常，当定义一个在线订购单时，我们可能需要允许用户输入两套地址信息：一个是送货目的地，另一个是账单目的地。大多数情况下，这两个地址是一样的，让用户两次输入相同的信息会降低用户友好度，这不是我们想要的。

可以编写服务器端代码来假定账单地址和送货地址相同（如果账单表单留空的话）。但是假设我们的产品经理有点偏执，要求用户明确确认这两个地址是一样的。

可以通过如下方式来满足产品经理的需求：向账单地址添加一个复选框来表明账单地址是否

与送货地址相同。如果选中复选框，会从送货地址字段复制内容到账单地址字段，并且设置账单地址为只读状态。取消勾选复选框会清空账单地址字段的值并取消只读状态。

图 7-1a 展示了状态改变前的测试表单，图 7-1b 展示了状态改变后的测试表单。

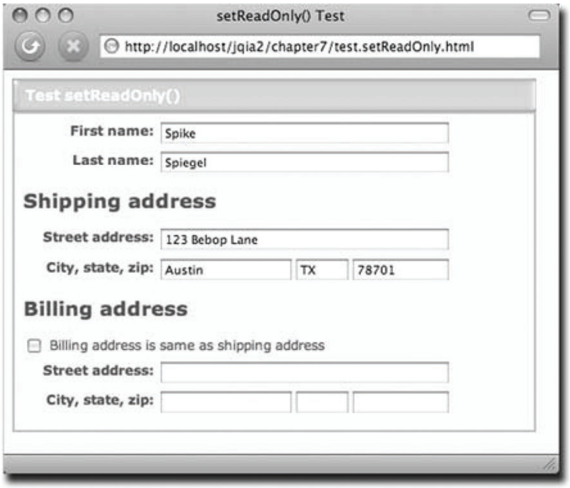


图 7-1a　测试 `setReadOnly()` 自定义包装器方法的表单，展示了选中复选框之前的状态

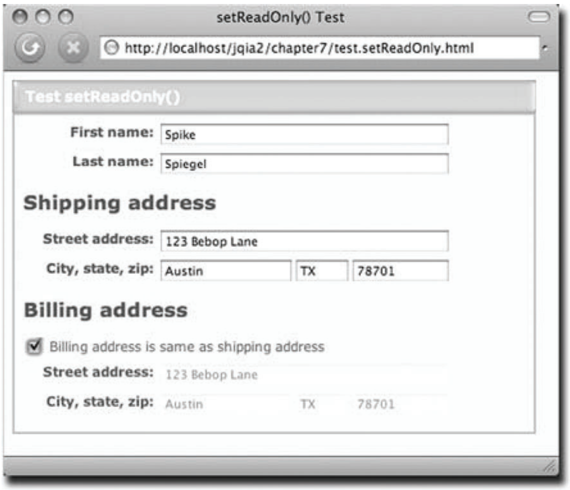


图 7-1b　测试 `setReadOnly()` 自定义包装器方法的表单，展示了选中复选框之后应用自定义方法的结果

这个测试表单的页面可以从文件 chapter7/test.setReadOnly.html 获取，代码如代码清单 7-4 所示。

代码清单 7-4　为 setReadOnly()包装器方法实现的测试页面

```
<!DOCTYPE html>
<html>
  <head>
    <title>setReadOnly() Test</title>
    <link rel="stylesheet" type="text/css" href="../styles/core.css" />
    <link rel="stylesheet" type="text/css" href="test.setReadOnly.css" />
    <script type="text/javascript" src="../scripts/jquery-1.4.js"></script>
  <script type="text/javascript" src="jquery.jqia.setReadOnly.js"></script>
  <script type="text/javascript">
    $(function(){
      $('#sameAddressControl').click(function(){
        var same = this.checked;
        $('#billAddress').val(same ? $('#shipAddress').val():'');
        $('#billCity').val(same ? $('#shipCity').val():'');
        $('#billState').val(same ? $('#shipState').val():'');
        $('#billZip').val(same ? $('#shipZip').val():'');
        $('#billingAddress input').setReadOnly(same);
      });
    });
  </script>
</head>

<body>
  <div class="module">
    <div class="banner">
      <img src="../images/module.left.cap.png" alt="" style="float:left"/>
      <img src="../images/module.right.cap.png" alt=""
           style="float:right"/>
      <h2>Test setReadOnly()</h2>
    </div>
    <div class="body">

      <form name="testForm">
        <div>
          <label>First name:</label>
          <input type="text" name="firstName" id="firstName"/>
        </div>
        <div>
          <label>Last name:</label>
          <input type="text" name="lastName" id="lastName"/>
        </div>
        <div id="shippingAddress">
          <h2>Shipping address</h2>
          <div>
            <label>Street address:</label>
            <input type="text" name="shipAddress" id="shipAddress"/>
          </div>
          <div>
            <label>City, state, zip:</label>
            <input type="text" name="shipCity" id="shipCity"/>
            <input type="text" name="shipState" id="shipState"/>
            <input type="text" name="shipZip" id="shipZip"/>
          </div>
```

```
          </div>
          <div id="billingAddress">
            <h2>Billing address</h2>
            <div>
              <input type="checkbox" id="sameAddressControl"/>
              Billing address is same as shipping address
            </div>
            <div>
              <label>Street address:</label>
              <input type="text" name="billAddress"
                     id="billAddress"/>
              </div>
              <div>
                <label>City, state, zip:</label>
                <input type="text" name="billCity" id="billCity"/>
                <input type="text" name="billState" id="billState"/>
                <input type="text" name="billZip" id="billZip"/>
              </div>
            </div>
          </form>
        </div>
      </div>

    </body>
  </html>
```

我们不会过多地讨论该页面的操作，因为这相当简单。这个页面真正令人感兴趣的地方只有一处：在就绪处理器中附加到复选框的单击处理器。当通过单击改变复选框的状态时，我们以下3件事情。

(1) 将选中状态复制到参数 `same`，以便在监听器的其余部分中引用。

(2) 设置账单地址字段的值。如果和送货地址相同，就将送货地址信息里的相应字段的值复制到账单地址字段；否则清空账单地址字段。

(3) 在所有在账单地址容器中的输入框上调用新的 `setReadOnly()`方法。

哎呀！我们在最后一步有点粗心了。使用`$('#billingAddress input')`创建的包装集不仅包含账单地址块里的文本字段，也包含复选框。复选框元素没有只读语义，但是它的不透明度可以被改变——但这绝不是我们的本意！

幸运的是，在定义插件的时候我们没有粗心，才化解了这个疏漏。还记得在 `setReadOnly()`方法中应用别的方法之前，就过滤掉了除文本字段以外的所有字段。我们非常赞同对细节的关注，尤其是对于打算提供给公众使用的插件。

可以从哪些方面对这个方法进行改进呢？考虑进行下列修改。

- 我们忘了文本区了！如何修改代码以便同时包含文本字段和文本区？
- 在两种状态[①]下应用到字段的不透明度水平都是被硬编码到函数中的。这对调用者不太友好。修改这个方法以便允许调用者提供不透明度水平。

7

① 只读状态或者非只读状态。

- 噢，糟糕，为什么要强迫开发者接受只能改变不透明度功能呢？如何修改这个方法让开发者可以决定字段在两种状态下的表现形式呢？

下面来看一个更加复杂的插件。

7.4.2　保留在包装器方法里的状态

每个人都喜欢幻灯片！

至少在 Web 上如此。不同于晚餐结束后的倒霉客人，被迫耐着性子看完令人头昏脑胀没完没了且是毫无重点的度假照片展，Web 幻灯片的访问者只要乐意就可以随时离开，不会伤害任何人的感情！

作为更复杂的插件示例，我们将会开发一个 jQuery 方法来允许 Web 开发者轻松地创建灵巧的幻灯片放映页面。我们将创建一个名为 Photomatic 的 jQuery 插件，然后生成测试页面来分步测试这个插件。完成后的测试页面如图 7-2 所示。



图 7-2　用来分步测试 `Photomatic` 插件的测试页面

这个页面运用了以下组件：

- 一组缩略图；
- 可以从缩略图列表里获取的一幅图片的全尺寸照片；
- 一组按钮用来手动播放幻灯片以及开始和停止自动播放幻灯片。

这个页面的行为如下。

- ❑ 单击任何缩略图就会显示相应的全尺寸图片。
- ❑ 单击全尺寸图片就会显示下一幅图片。
- ❑ 单击按钮就会执行相应的操作：
  - First——显示第一幅图片；
  - Previous——显示上一幅图片；
  - Next——显示下一幅图片；
  - Last——显示最后一幅图片；
  - Play——开始自动播放照片直到再次单击。
- ❑ 任何经过图片列表末尾的操作都会折回到另一端。在最后一幅图片出现时单击 Next 按钮就会显示第一幅图片，反之亦然。

我们也想赋予 Web 开发者尽可能多的自由来布局页面和设计样式。我们将定义插件，以便开发者能够以喜欢的方式来创建元素，然后告诉我们哪个页面元素应该用于哪个目的。不止如此，为了给 Web 开发者尽可能多的回旋余地，我们将定义插件，以便开发者能够提供任何图片包装集作为缩略图列表。通常，缩略图会集中在一起，就像测试页面那样，但是开发者可以自由地把页面上的任何图片当作缩略图。

首先介绍 Photomatic 插件的语法。

方法语法：photomatic

photomatic(options)

设置缩略图包装集以及在 options 散列对象里标识的页面元素，将它们作为 Photomatic 的控件进行操作

参数

options　　（对象）指定 photomatic 的各种设置的散列对象。参见表 7-1 以了解细节

返回值

包装集

因为要用到许多重要的参数来控制 Photomatic 的操作（其中很多参数可以省略），所以使用散列对象来传递这些参数，我们曾在 7.2.3 节中讨论过。可以指定的选项如表 7-1 所示。

表7-1　Photomatic自定义插件方法的选项

选项名称	描　　述
firstControl	（选择器|元素）作为First控件的元素引用或用来标识DOM元素的jQuery选择器。如果省略，则不设置控件
lastControl	（选择器|元素）作为Last控件的元素引用或用来标识DOM元素的jQuery选择器。如果省略，则不设置控件

（续）

选项名称	描　　述
nextControl	（选择器|元素）作为Next控件的元素引用或用来标识DOM元素的jQuery选择器。如果省略，则不设置控件
photoElement	（选择器|元素）作为全尺寸照片显示的元素引用或用来标识<img>元素的jQuery选择器。如果省略，则默认为jQuery选择器img.photomaticPhoto
playControl	（选择器|元素）作为Play控件的元素引用或用来标识DOM元素的jQuery选择器。如果省略，则不设置控件
previousControl	（选择器|元素）作为Previous控件的元素引用或用来标识DOM元素的jQuery选择器。如果省略，则不设置控件
transformer	（函数）用来将缩略图片的URL转换为相应的全尺寸图片的URL。如果省略，则默认将URL中所有的thumbnail替换为photo
delay	（数字）幻灯片自动播放时切换照片的间隔时间（以毫秒为单位）。默认为3 000

出于对“测试驱动开发”理念的认同，在着手创建 Photomatic 插件之前，先来为这个插件创建测试页面。该页面的代码如代码清单 7-5 所示，可以从文件 chapter7/photomatic/photomatic.html 获取。

代码清单 7-5　用来创建如图 7-2 所示的 Photomatic 的测试页面

```
<!DOCTYPE html>
<html>
  <head>
    <title>Photomatic Test</title>
    <link rel="stylesheet" type="text/css"
          href="../../styles/core.css">
    <link rel="stylesheet" type="text/css" href="photomatic.css">
    <script type="text/javascript"
            src="../../scripts/jquery-1.4.js"></script>
    <script type="text/javascript"
            src="jquery.jqia.photomatic.js"></script>
    <script type="text/javascript">
      $(function(){                                        ❶ 调用 Photomatic 插件
        $('#thumbnailsPane img').photomatic({             <┘
          photoElement: '#photoDisplay',
          previousControl: '#previousButton',
          nextControl: '#nextButton',
          firstControl: '#firstButton',
          lastControl: '#lastButton',
          playControl: '#playButton',
          delay: 1000
        });
      });
    </script>
  </head>
  <body class="fancy">

    <div id="pageContainer">
      <div id="pageContent">
```

```
      <h1>Photomatic Tester</h1>                       ❷ 包含缩略图

      <div id="thumbnailsPane">                      <┘
        <img src="thumbnails/IMG_2212.jpg"/>
        <img src="thumbnails/IMG_2222.jpg"/>
        <img src="thumbnails/IMG_2227.jpg"/>
        <img src="thumbnails/IMG_2235.jpg"/>
        <img src="thumbnails/IMG_2259.jpg"/>
        <img src="thumbnails/IMG_2269.jpg"/>
        <img src="thumbnails/IMG_2273.jpg"/>
        <img src="thumbnails/IMG_2287.jpg"/>
        <img src="thumbnails/IMG_2292.jpg"/>
        <img src="thumbnails/IMG_2296.jpg"/>
        <img src="thumbnails/IMG_2298.jpg"/>
        <img src="thumbnails/IMG_2302.jpg"/>
        <img src="thumbnails/IMG_2310.jpg"/>
        <img src="thumbnails/IMG_2319.jpg"/>
        <img src="thumbnails/IMG_2331.jpg"/>
        <img src="thumbnails/IMG_2335.jpg"/>
      </div>
                                                           ❸ 定义全尺寸照片图片元素
      <div id="photoPane">
        <img id="photoDisplay" src=""/>                  <┘
      </div>
                                                  ❹ 包含作为控件的元素
      <div id="buttonBar">                        <┘
        <img src="button.placeholder.png" id="firstButton"
             alt="First" title="First photo"/>
        <img src="button.placeholder.png" id="previousButton"
             alt="Previous" title="Previous photo"/>
        <img src="button.placeholder.png" id="playButton"
             alt="Play/Pause" title="Play or pause slideshow"/>
        <img src="button.placeholder.png" id="nextButton"
             alt="Next" title="Next photo"/>
        <img src="button.placeholder.png" id="lastButton"
             alt="Last" title="Last photo"/>
      </div>

    </div>
  </div>

</body>
</html>
```

如果这看起来比你认为的简单，也不用惊讶。通过应用不唐突的 JavaScript 原则并将所有的样式信息保存在外部样式表里，标记整齐而简单。事实上，页面上的脚本由调用插件的单个语句❶组成，内存占用很少。

HTML 标记由保存缩略图的容器❷、保存全尺寸照片的图片元素（初始 src 特性为空）❸，以及控制幻灯片播放的一组按钮❹组成。其他事情都由新插件来处理。

下面就来开发新插件吧。

首先搭建框架（随后会逐步填充细节）。目前的起点应该看起来很熟悉，因为它遵循我们一直以来使用的模式。

```
(function($){
  $.fn.photomatic = function(options) {
  };
})(jQuery);
```

这段代码定义初始为空的包装器函数，它接受名为 options 的单个散列参数（和语法描述的一样）。首先，在函数主体的内部，我们把这些调用者设置的参数与表 7-1 所描述的默认设置合并起来，得到一个可以在方法的剩余代码中引用的单个 settings 对象。

使用下列的习惯用法来执行这个合并操作（我们已经看到过几次了）：

```
var settings = $.extend({
      photoElement: 'img.photomaticPhoto',
      transformer: function(name) {
                     return name.replace(/thumbnail/,'photo');
                   },
      nextControl: null,
      previousControl: null,
      firstControl: null,
      lastControl: null,
      playControl: null,
      delay: 3000
    },options||{});
```

在这个语句执行之后，settings 变量将会包含内联散列对象提供的默认值，这个默认值会被调用者提供的值所覆盖。虽然没必要包含无默认值的属性（值为 null 的属性），但是为了文档编写的目的，我们觉得列出所有可能的设置是有用和明智的。

还有几件事情需要跟踪。为了让控件知道“下一幅”图片和“上一幅”图片的意思，不仅需要一个由缩略图组成的有序列表，还需要一个用来找出“当前显示”图片的指示器。

缩略图列表就是这个方法正在（或者至少应该是）操作的对象。我们不知道开发者在包装集里收集了什么，因此想筛选出图片元素，这个操作可以很容易地通过 jQuery 选择器来实现。但是应该在哪里存储这个列表呢？

可以创建另一个变量来保存缩略图列表，但是集中存储可能会更好。下面将该列表作为另一个属性保存到 settings 上，如下所示：

```
settings.thumbnails$ = this.filter('img');
```

从包装集中过滤图片元素（在方法内可以通过 this 访问）来生成一个新包装集（只包含 <img>元素），这个新包装集被存储在 settings 的一个名为 thumbnails$的属性中（尾部的$符号是一种约定的写法，用来表示这是一个对包装集的存储引用）。

另一个需要跟踪的状态是当前图片。通过给 settings 添加另一个名为 current 的属性来保持缩略图列表的下标，以便跟踪当前图片：

```
settings.current = 0;
```

关于缩略图还有一个步骤需要完成。如果通过跟踪下标来跟踪当前的照片，那么至少还有一种情况（我们很快就会考察到），即在给予缩略图元素引用的情况下，我们需要知道该缩略图元素的下标。最简单的处理办法是预先考虑这个需求，并使用便捷的 jQuery 的 data()方法在每个缩略图元素上记录缩略图的索引。使用以下语句可以完成这一任务：

```
settings.thumbnails$
  .each(
    function(n){ $(this).data('photomatic-index',n); }
  )
```

这个语句遍历每一个缩略图元素，给每个图片添加名为 photomatic-index 的数据元素来记录其在列表里的下标。既然已经设置了初始状态，我们就可以朝着插件的核心内容前进——设置控件、缩略图以及照片显示。

等一下！初始状态？我们怎么能够期望在就要结束执行的函数内使用本地变量来跟踪状态呢？当函数返回时，这个变量和所有的设置不都超出作用域了吗？

通常这可能是正确的，但是还有一种情况，当变量作为闭包的一部分的时候，这种变量会比在通常的作用域中保存得更久。之前已经了解过闭包，不过如果你仍然对闭包心存疑虑，请复习附录的内容。你必须理解闭包，不仅是为了完成 Photomatic 插件的实现，也是为了创建一些至关重要的插件。

在考虑剩余的工作时，我们认识到需要附加大量事件监听器到控件和元素上，目前为止为了识别这些控件和元素我们煞费苦心。因为当声明监听器函数的时候 settings 变量是在作用域内的，所以每个监听器将会成为包含 settings 变量的闭包的一部分。因此可以放心，即使 settings 变量看起来可能是暂时存在的，但是它所代表的状态将会一直存在并且可供所有定义的监听器使用。

说到监听器，下面列出了需要附加到不同元素上的 click 事件监听器。

- 单击一张缩略图照片将会导致其全尺寸版本显示出来。
- 单击全尺寸照片将会显示下一幅照片。
- 单击定义为 Previous 控件的元素将会显示上一幅图片。
- 单击 Next 控件将会显示下一幅图片。
- 单击 First 控件将会显示列表里的第一幅图片。
- 单击 Last 控件将会显示列表里的最后一幅图片。
- 单击 Play 控件将会开始自动播放幻灯片，播放时使用在设置里定义的延迟时间。随后再次单击控件会停止播放幻灯片。

观察这个列表，我们马上就会注意到所有的这些监听器都有一个共同点：它们都必须显示缩略图对应的全尺寸照片。作为优秀而聪明的程序员，我们想把上述共同的处理提取出来封装到一个函数里，这样就不用一遍又一遍地重复编写同样的代码了。

但是如何实现呢？

如果在编写常规的页面上的 JavaScript 代码，那就可以定义顶级函数；如果编写的是面向对象的 JavaScript，那就可以在 JavaScript 对象上定义方法。但是我们编写的是一个 jQuery 插件。应该在哪里定义实现函数呢？

我们不想为了只应该从插件代码内部调用的函数而侵占全局命名空间或$命名空间，那能做什么呢？噢，为目前的难题再添加一个需求，让函数参与到包含 settings 变量的闭包中，这样就不用将 settings 变量作为参数传递到每次的调用中了。

JavaScript 作为一种函数式语言的能力再次拯救了我们，它允许我们在插件函数的内部定义这个新函数。通过这样做，我们将这个新函数的作用域限制在插件函数内部（我们的目标之一），并且因为settings变量处于这个作用域内，所以它与新函数形成了一个闭包（我们的另一目标）。难道还有更简单的办法呢？

因此我们在插件函数内部定义一个名为 showPhoto()的函数，它接受一个指示缩略图（它的全尺寸照片将被显示）下标的单个参数，如下所示：

```
function showPhoto(index) {
  $(settings.photoElement)
    .attr('src',
          settings.transformer(settings.thumbnails$[index].src));
  settings.current = index;
};
```

当传入缩略图的 index 时（其相应的全尺寸照片将被显示），这个新函数会使用 settings 对象里的值完成下述操作（settings 对象可以从通过函数声明创建的闭包里获取）：

(1) 查找通过 index 识别的缩略图的 src 特性；

(2) 传递 src 特性值到 transformer 函数来把它从缩略图 URL 转换到全尺寸图 URL；

(3) 把转换结果赋值给全尺寸图片元素的 src 特性；

(4) 记录被显示照片的下标为新的当前下标。

有了这个便捷的函数，我们就可以定义前面列出的监听器了。下面从设置缩略图开始，只需要能够引发缩略图相应的全尺寸照片显示出来即可。把对 click()函数调用的语句和前面引用 settings.thumbnails$的语句链接起来，如下所示：

```
.click(function(){
  showPhoto($(this).data('photomatic-index'));
});
```

在这个处理器中，我们获取了缩略图下标的值（已经保存到 photomatic-index 数据元素里），并使用这个值来调用 showPhoto()函数。这个处理器证明了之前编写的设置是非常成功的！

设置照片显示元素来显示列表里的下一张照片也很简单：

```
$(settings.photoElement)
  .attr('title', 'Click for next photo')
  .css('cursor','pointer')
  .click(function(){
    showPhoto((settings.current+1) % settings.thumbnails$.length);
  });
```

我们特意在照片上添加了一个 title 特性，这样用户就知道单击当前照片将会播放下一张照片，并且还设置了光标用来指示元素是可单击的。

然后创建一个单击处理器，在其中用下一个下标值来调用 showPhoto()函数——注意我们是如何使用 JavaScript 取模操作符（%）来在超出列表长度的时候回到列表的开始位置。

First、Previous、Next 和 Last 控件的处理器都遵循类似的模式：找出合适的缩略图（其对应的全尺寸照片将会被显示）下标，然后用该下标来调用 showPhoto()：

```
$(settings.nextControl).click(function(){
  showPhoto((settings.current+1) % settings.thumbnails$.length);
});
$(settings.previousControl).click(function(){
  showPhoto((settings.thumbnails$.length+settings.current-1) %
            settings.thumbnails$.length);
});
$(settings.firstControl).click(function(){
  showPhoto(0);
});
$(settings.lastControl).click(function(){
  showPhoto(settings.thumbnails$.length-1);
});
```

Play 控件的设置更加复杂一些。这个控件必须启动一个遍历整个照片集的进程，而不是显示一张特定的照片，再次单击时会停止这个进程。下面来看一下用来实现这个过程的代码：

```
$(settings.playControl).toggle(
  function(event){
    settings.tick = window.setInterval(
      function(){
        $(settings.nextControl).triggerHandler('click')
      },
      settings.delay);
    $(event.target).addClass('photomatic-playing');
    $(settings.nextControl).click();
  },
  function(event){
    window.clearInterval(settings.tick);
    $(event.target).removeClass('photomatic-playing');
  }
);
```


首先，注意所使用的 jQuery 的 `toggle()`方法在每隔一次单击控件时可以在两个不同的监听器之间切换。这样就避免了自己确认是该开始播放还是停止播放幻灯片。

在第一个处理器中，我们使用 JavaScript 提供的 `setInterval()`方法使一个函数在 `delay` 间隔中持续触发。我们将间隔计时器对象保存在 `settings` 变量中供以后引用。

我们也给 Play 控件添加了类 `photomatic-playing`，这样 Web 开发者可以使用 CSS 来改变其样式了（如果需要的话）。

作为处理器里的最后一个动作，我们在 Next 控件上模拟了一个单击来立即推进到下一张照片（而不是要等到第一个时间间隔结束）。

在 `toggle()`调用的第二个处理器中，我们想要停止幻灯片播放，因此使用 `clearInterval()` 清空了间隔计时器对象并且从 Play 控件上删除了类 `photomatic-playing`。

我打赌你不会想到它是这么简单。

在宣布成功之前，还有两个最后的任务：我们需要在任何用户操作前显示第一张照片，而且还需要返回原始的包装集以便我们的插件可以参与到 jQuery 方法链中。可以使用以下代码完成上述任务：

```
showPhoto(0);
return this;
```

跳个简短的庆功舞吧，我们终于完成了！

完整的插件代码如代码清单 7-6 所示，也可以从文件 chapter7/photomatic/jquery.jqia.photomatic.js 获取。

代码清单 7-6　Photomatic 插件的完整实现

```
(function($){

  $.fn.photomatic = function(options) {
    var settings = $.extend({
      photoElement: 'img.photomaticPhoto',
      transformer: function(name) {
                     return name.replace(/thumbnail/,'photo');
                   },
      nextControl: null,
      previousControl: null,
      firstControl: null,
      lastControl: null,
      playControl: null,
      delay: 3000
    },options||{});
    function showPhoto(index) {
      $(settings.photoElement)
        .attr('src',
              settings.transformer(settings.thumbnails$[index].src));
      settings.current = index;
    }
    settings.current = 0;
    settings.thumbnails$ = this.filter('img');
    settings.thumbnails$
      .each(
        function(n){ $(this).data('photomatic-index',n); }
      )
      .click(function(){
        showPhoto($(this).data('photomatic-index'));
      });
    $(settings.photoElement)
      .attr('title','Click for next photo')
      .css('cursor','pointer')
      .click(function(){
        showPhoto((settings.current+1) % settings.thumbnails$.length);
      });
    $(settings.nextControl).click(function(){
      showPhoto((settings.current+1) % settings.thumbnails$.length);
    });
    $(settings.previousControl).click(function(){
      showPhoto((settings.thumbnails$.length+settings.current-1) %
                settings.thumbnails$.length);
    });
    $(settings.firstControl).click(function(){
      showPhoto(0);
    });
    $(settings.lastControl).click(function(){
```

```
      showPhoto(settings.thumbnails$.length-1);
    });
    $(settings.playControl).toggle(
      function(event){
        settings.tick = window.setInterval(
          function(){ $(settings.nextControl).triggerHandler('click'); },
          settings.delay);
        $(event.target).addClass('photomatic-playing');
        $(settings.nextControl).click();
      },
      function(event){
        window.clearInterval(settings.tick);
        $(event.target).removeClass('photomatic-playing');
      });
    showPhoto(0);
    return this;
  };

})(jQuery);
```

这使用 jQuery 代码的典型插件，它在简洁的代码里包含了强大的功能。它展示了一套重要的技巧——使用闭包跨越 jQuery 插件的作用域来维持状态，以及启用可以由插件来定义和使用的私有实现函数的创建，而无需侵占任何命名空间。

另外请注意，因为我们没有让状态从插件中“泄漏”出去，所以可以在页面上自由地添加任意数量的 `Photomatic` 部件，而不用担心这些部件互相干扰（当然也要小心，不要在标记里使用重复的 `id` 值）。

但这就算完成了吗？这由你决定，也可以思考以下练习。

- 再次强调，错误检测和处理没有被包含其中。你应该如何做来让这个插件尽可能地无懈可击？
- 照片之间的过渡是瞬间发生的。利用你在第 5 章中学到的知识来修改插件，让照片以淡入淡出的方式过渡到下一张。
- 更进一步，你应该如何让开发者使用自定义动画呢？
- 为了最大限度的灵活性，我们编写了这个插件来设置已经被用户创建的 HTML 元素。如何创建一个类似的插件（但拥有更少的显示自由度）来动态生成所有需要的 HTML 元素呢？

你现在已经整装待发，可以去编写自己的 jQuery 插件。如果发现了一些有用的插件，可以与 jQuery 社区的其他用户分享。访问 http://plugins.jquery.com/以获取更多信息。

7.5 小结

本章介绍了如何编写用来扩展 jQuery 的可重用代码。

把自己的代码编写为 jQuery 扩展有许多优点。这不仅可以在 Web 应用中保持代码的一致性（无论使用的是 jQuery API 还是自定义 API），还可以使我们的代码充分利用 jQuery 提供的所有强大功能。

遵循一些命名规则可以帮助避免文件名之间的命名冲突，也可以避免页面重新指定$时可能会产生的问题。

创建新的实用函数就像在$上创建新的函数属性一样简单，而新的包装器方法可以简单地创建为$.fn 的属性。

如果插件创作激发了你的兴趣，我们强烈建议你去下载和熟悉已有的插件代码，看看插件作者们是如何实现插件功能的。你将会看到本章所阐述的技巧是如何在插件代码中被广泛使用的，并且会学到本书范围外的新技巧。

还有更多的 jQuery 知识可供运用，接下来开始学习 jQuery 是如何把 Ajax 合并到交互应用中，就像“不用脑子”那样简单。

第 8 章

使用 Ajax 与服务器通信

本章内容

- ❑ Ajax 概述
- ❑ 从服务器加载预定义格式的 HTML
- ❑ 发起常规的 GET 和 POST 请求
- ❑ 对请求进行细粒度的控制
- ❑ 设置默认的 Ajax 属性
- ❑ 处理 Ajax 事件

没有哪种单项的技术手段比 Ajax 更能改变 Web 的外观，这是毫无争议的。无需重新加载整个页面就可以向服务器发起异步请求的能力，开启了一整套完全崭新的用户交互范例并且使得 DOM 脚本的应用成为可能。

Ajax 添加到 Web 工具箱的时间比很多人认为的要早。早在 1998 年，微软就引入了一个 ActiveX 控件，从而能够在脚本控制下执行异步请求（不考虑使用<iframe>元素实现这种活动的情况），作为所创建 OWA[①]（Outlook Web Access）的一部分。尽管 OWA 取得了一定程度的成功，但是似乎很少有人关注其中潜在的技术。

8

几年过去了，一些事件使得 Ajax 进入了 Web 开发社区的集体意识。几个非微软浏览器将该技术的标准化版本实现为 XMLHttpRequest（XHR）对象。Google 开始使用 XHR。在 2005 年，Adaptive Path 的 Jesse James Garrett 创造了术语 Ajax（Asynchronous JavaScript and XML，异步的 JavaScript 与 XML）。

好像只是为了等待给这门技术赋予一个吸引人的名称，Ajax 迅速地赢得了 Web 开发人员的广泛关注，成为了开启 DOM 脚本应用主要工具之一。

本章，我们将简要浏览 Ajax（如果你已经是 Ajax 专家，可以跳到 8.2 节），然后看看 jQuery 如何让使用 Ajax 易如反掌。

下面先回顾一下 Ajax 技术到底是什么样的技术。

① 通过 Web 浏览器（而非 Outlook 客户端）来访问微软 Exchange Server 邮箱的方式。

8.1　回顾 Ajax

尽管我们会在本节中快速浏览 Ajax，但请注意，这节内容不能作为一个完整的 Ajax 指南或者 Ajax 初级读本。如果你完全不熟悉 Ajax（甚至更糟，认为我们在谈论一种洗洁精或者一个虚构的希腊英雄），我希望你学习关于 Ajax 的一些资源来熟悉这门技术。Manning 的图书 *Ajax in Action* 和 *Ajax in Practice*[①]都是优秀的范例。

有些人可能会争论说，术语 Ajax 指的是无需刷新面向用户的页面就能发出服务器请求的所有方法（例如向隐藏的`<iframe>`元素提交请求），但是大部分人将这个术语与使用 `XMLHttpRequest`（XHR）或者微软的 XMLHTTP ActiveX 控件的行为联系在一起。

总体流程图如图 8-1 中所示，我们将每次考察其中的一个步骤。

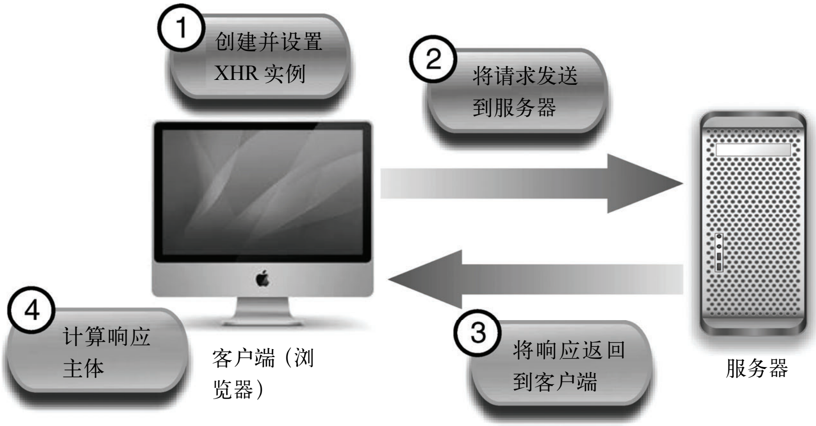


图 8-1　Ajax 请求的生命周期，从客户端向服务器端发送请求，然后再从服务器端返回

下面先来创建一个 XHR 实例，看看如何使用这些对象生成对服务器的请求。

8.1.1　创建XHR实例

在完美的世界中，为一个浏览器编写的代码可以在所有常用的浏览下工作。我们已经知道自己并非生活在完美世界中，即便有了 Ajax，情况也是一样。通过 JavaScript 的 XHR 对象来发起异步请求有一个标准方法，而且 IE 浏览器也有使用 ActiveX 控件的专有方式。在 IE7 下可以使用一个模仿标准接口的包装器，但是 IE6 却需要不同的代码。

注意　jQuery 的 Ajax 实现（将贯穿本章的其余部分）没有使用 IE 浏览器的包装器，因为不正确的实现会导致引用问题。相反，jQuery 在 ActiveX 对象可用的时候使用它。这对我们来说是好消息！通过使用 jQuery 来满足对 Ajax 需求，就可以知道最好的实践已经被研究出来并将被应用到实际。

① 这两本书的中文版分别为《Ajax 实战》和《Ajax 实战：实例详解》，已由人民邮电出版社出版。——编者注

一旦创建了 XHR 对象，用来建立、初始化以及响应请求的代码就是相对于浏览器而独立的，并且为任何特殊的浏览器创建 XHR 实例都很容易。问题是不同的浏览器实现 XHR 的方式都不一样，我们需要按照适合当前浏览器的方式来创建 XHR 实例。

我们将使用在第 6 章中介绍的推荐技术特征检测，而不是检测用户运行的浏览器来决定采用哪种方式。使用这种技术，我们尝试计算出浏览器的特征而不是分析使用的什么浏览器。特征检测会使代码更加稳健，因为它可以在支持所测试特征的任何浏览器中工作。

代码清单 8-1 展示了一个典型的使用该技术实例化 XHR 的习惯用法。

代码清单 8-1　功能检测使得可以在很多浏览器中使用 Ajax 的代码

```
var xhr;
if (window.ActiveXObject) {                         // 测试 ActiveX 是否存在
  xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
else  if (window.XMLHttpRequest) {                  // 测试 XHR 是否已经被定义
  xhr = new XMLHttpRequest();
}
else {                                              // 如果不支持 Ajax 则抛出错误
  throw new Error("Ajax is not supported by this browser");
}
```

在 XHR 实例被创建之后，它拥有一组方便一致的属性和方法，可以跨所有支持的浏览器实例。这些属性和方法如表 8-1 所示，其中最常用的属性和方法将在接下来的几节中讨论。

表8-1　**XMLHttpRequest**（XHR）方法和属性

方　法	描　述
abort()	导致当前正在执行的请求被取消
getAllResponseHeaders()	返回包含所有响应头的名称和值的单个字符串
getResponseHeader(name)	返回响应头中指定名称的值
open(method, url, async, username, password)	设置HTTP方法（GET或者POST）和请求的目标URL。可以声明为同步请求，也可以为需要基于容器认证的请求提供用户名和密码（可选）
send(content)	发出带有指定主体内容（可选）的请求
setResuestHeader(name, value)	使用指定的名称和值设置请求头
属　性	**描　述**
onreadystatechange	当请求状态改变时要调用的事件处理器
readyState	一个指示活动请求的当前状态的整数值，如下所示： 0 = UNSENT 1 = OPENED 2 = HEADERS_RECEIVED 3 = LOADING 4 = DONE
responseText	在响应中返回的主体内容

8

（续）

方　　法	描　　述
responseXML	如果主体内容被识别为XML，则从主体内容创建XML DOM
status	从服务器返回的响应状态码。例如：200表示成功，404表示未找到。查阅HTTP规范[a]来可以获取完整的状态码列表
statusText	响应返回的状态文本信息

a HTTP 1.1 状态码的定义来自文档 RFC 2616: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10。

注意　想从一个权威可靠的来源获取此信息吗？XHR 规范可以在这里找到：http://www.w3.org/TR/XMLHttpRequest/。

现在我们已经得到了一个已创建的 XHR 实例，下面就来看下建立和发起到服务器的请求需要哪些步骤。

8.1.2　发起请求

在向服务器发出请求之前，需要执行如下的设置步骤：

(1) 指定 HTTP 方法，比如（POST 或者 GET）；

(2) 提供要接触的服务器端资源的 URL；

(3) 告诉 XHR 实例如何通报进度；

(4) 为 POST 请求提供任何主体内容。

通过调用 XHR 的 open()方法来设置前两项，如下所示：

```
xhr.open('GET','/some/resource/url');
```

注意，这个方法不会将请求发送到服务器。它只不过是设置了要使用的 URL 和 HTTP 方法。也可以向 open()方法传递第三个布尔型参数，将请求指定为异步的（如果参数值为 true，这也是默认值）还是同步的（如果参数值为 false）。发起同步请求的需要不是很多（虽然这样做意味着无需处理回调函数）。毕竟，请求的异步本质通常是以异步方式来发起请求的理由。

在第三步中，必须提供一种方式让 XHR 实例通知我们正在发生什么。通过将一个回调函数赋值给 XHR 对象的 onreadystatechange 属性来达到这个目的。XHR 实例会在其处理进程的各个阶段调用这个被称为*就绪状态处理器*的函数。通过查看 XHR 其他属性的设置，就可以确切地知道请求中正在发生什么。我们将在下一节学习一个典型的就绪状态处理器是如何运行的。

发起请求的最后步骤是为 POST 请求提供主体内容并把它发送到服务器。这些步骤都是通过 send()方法来完成的。对于通常没有主体的 GET 请求，就不需要传递主体内容参数，如下所示：

```
xhr.send(null);
```

当向 POST 请求传递请求参数时，传入 send()方法的字符串必须拥有正确的格式（可以看

作是查询字符串格式），其中名称和值都被正确地进行 URI 编码。URI 编码超出了本节的范围（事实上，jQuery 将为我们处理好这一切），但是如果你很好奇，可以搜索一下术语 encodeURI-Component，你会得到想要的东西。

这种调用的另一个示例如下：

```
xhr.send('a=1&b=2&c=3');
```

下面来看看到底什么是就绪状态处理器。

8.1.3 保持跟踪进度

XHR 实例通过就绪状态处理器通知我们它的进度。通过将作为就绪处理器的函数引用赋值给 XHR 实例的 onreadystatechange 属性来建立处理器。

一旦通过 send()方法发出请求，随着请求在其不同状态之间转变这个回调将会被调用多次。请求的当前状态可以通过 readyState 属性的数字代码来访问（在表 8-1 中可查看关于这个属性的描述）。

那太好了，但更多情况下，我们只对请求在什么时间完成以及它是否成功感兴趣。就绪处理器经常使用代码清单 8-2 中所示的习惯用法来实现。

代码清单 8-2　就绪状态处理器，忽略除 DONE 状态之外的所有状态

```
xhr.onreadystatechange = function() {
  if (this.readyState == 4) {          // 忽略除 DONE 状态之外的所有状态
    if (this.status >= 200 &&          // 根据响应状态进行代码分支
        this.status < 300) {
      //成功                            // 成功时执行
    }
    else {
      //出错                            // 失败时执行
    }
  }
}
```

这段代码忽略除 DONE 状态之外的所有状态，一旦检测出 DONE 状态，就检查 status 属性的值以确定请求是否成功。HTTP 规范定义所有位于 200 ~ 299 区间内的状态码为成功，等于或者大于 300 的状态码为各种类型的失败。

下面来探索如何处理已完成请求的响应。

8.1.4 获取响应

在就绪处理器确定了 readyState 处于完成状态并且请求已经成功完成之后，就可以从 XHR 实例中获取响应的主体了。

尽管 Ajax 的名称如此（其中 X 代表 XML），但是响应主体的格式可以是任何文本格式，并不局限于 XML。事实上，大多数时候，Ajax 请求的响应不是 XML 的格式。它可以是普通的文本，或者可能是 HTML 片段，甚至可以是使用 JSON（JavaScript Object Notation）格式（作为一

8

种交换格式正在变得越来越流行）的 JavaScript 对象或者数组的一段文本表示。

无论响应主体的格式如何，都可以通过 XHR 实例的 `responseText` 属性来获取响应主体（假设请求成功完成）。如果响应通过包含内容类型头（`Content-Type`，指定 MIME 类型为 `text/xml`、`application/xml`，或者 MIME 类型以`+xml` 结尾）来表明它的主体格式是 XML，那么响应主体将会解析为 XML。结果 DOM 将可以在 `responseXML` 属性中获取。然后可以使用 JavaScript（jQuery 则使用其选择器 API）来处理 XML DOM。

在客户端处理 XML 不是一件复杂的事情，但是即使在 jQuery 帮助下这仍然是一件痛苦的事情。尽管有时只能使用 XML 返回复杂的层次数据，但是当不需要使用 XML 的全部功能（会带来相应的麻烦）时，页面开发者经常会使用其他格式。

但是使用这些其他的格式也有麻烦。如果返回 JSON，它必须被转换为与其运行时等价的形式。如果返回 HTML，它必须被加载到合适的目标元素中。如果返回的 HTML 标记中包含需要求值的`<script>`块该怎么办？本节中不会涉及这些问题，因为本节不是一个完整的 Ajax 参考，而且更重要的是，因为 jQuery 已经帮助我们处理了其中大部分的问题。

此时，你可以回顾一下在图 8-1 中展示的整个流程。

在这个简短的 Ajax 概述中，我们找出了页面开发者在使用 Ajax 时需要处理的一些难点，如下所示：

- 实例化 XHR 对象需要特定于浏览器的代码；
- 就绪处理器需要处理很多乏味的的状态改变；
- 需要使用多种方式来处理响应主体，处理的方式取决于其格式。

本章其余的部分将描述 jQuery 的 Ajax 方法和实用函数如何使得在页面上使用 Ajax 更加容易（以及更加清晰）。jQuery 的 Ajax API 有多种选择，下面从一些最简单、最常用的工具开始。

8.2　加载内容到元素中

Ajax 的一个最常见的用途可能是从服务器抓取一堆内容并将其填充到 DOM 中的某个关键位置。这些内容可以是一个将要成为目标容器元素子内容的 HTML 片段，也可以是将要成为目标元素内容的普通文本。

为建立示例做好准备

与目前为止我们在本书中已经考察过的所有示例代码都不同，本章的代码示例需要 Web 服务器提供的服务以便接受对服务器端资源的 Ajax 请求。因为对服务器端操作机制的探讨超出了本书的范围，所以我们只建立一些很小的服务器端资源来把数据送回客户端，而无需操心真正的操作，只需将服务器看作一个“黑盒子”，我们不需要也不想知道它是如何完成工作的。

为了启用这些“黑盒”资源的服务，需要建立某种类型的 Web 服务器。为了方便起见，服务器端资源已经通过两种方式建立：JSP（Java Server Page）和 PHP。如果你正在运行（或者希望运行）一个 servlet/JSP 引擎，可以使用 JSP 资源；如果你想为 Web 服务器启用 PHP，可以使用 PHP 资源。

如果你想使用 JSP 资源但是没有正在运行的合适的服务器，可以参见包含在本章的示例代码中关于建立免费的 Tomcat Web 服务器的说明。你会在 chapter8/tomcat.pdf 文件中找到这些说明。不用担心，即使从未看过一行 Java 代码，建立 Tomcat 服务器比你预想的要简单得多！

下载的代码中的示例可以使用 JSP 或者 PHP 资源来建立，这取决于所建立服务器。

一旦建立了所选择的服务器，就可以测试 URL：http://localhost:8080/jqia2/chapter8/test.jsp（用来检测 Tomcat 的安装是否正确）或者 http://localhost/jqia2/chapter8/test.jsp（用来检测 PHP 的安装是否正确）。后者假设你已经建立了 Web 服务器（Apache 或者所选的任何其他服务器）来使用示例代码根文件夹作为文档基础。

当你成功地看到了相应的测试页面时，就可以准备运行本章中的示例了。

如果你不想在本地运行这些示例，也可以从这里远程运行示例代码：http://bibeault.org/jqia2。

假设在页面加载时，我们想使用名为 `someResource` 的资源从服务器获取一大段 HTML，并把它作为 `id` 为 `someContainer` 的`<div>`元素的内容。这是在本章中最后一次说明如何在没有 jQuery 帮助的情况下完成这个任务。使用我们在本章早些时候建立的模式，`onload` 处理器的主体代码如代码清单 8-3 所示。这个示例的完整 HTML 文件可以在文件 chapter8/listing.8.3.html 中找到。

注意 必须再次使用 Web 服务器来运行这个示例（不能只是在浏览器中打开文件），因此 URL 应该是 http://localhost:8080/jqia2/chapter8/listing.8.3.html。如果使用 Apache 则忽略端口规范:8080，如果使用 Tomcat 则保留此端口。

本章后面出现的 URL，我们将使用符号`[:8080]`来表明是否需要端口号，但是一定不要在 URL 中包含中括号。

代码清单 8-3 使用原生 XHR 来获取并包含 HTML 片段

```
var xhr;

if(window.ActiveXObject) {
  xhr = new ActiveXObject("Microsoft.XMLHTTP");
   }
else if (window.XMLHttpRequest) {
  xhr = new XMLHttpRequest();
}
else {
  throw new Error("Ajax is not supported by this browser");
}

xhr.onreadystatechange = function() {
  if (this.readyState == 4) {
    if (this.status >= 200 && this.status < 300) {
      document.getElementById('someContainer')
        .innerHTML = this.responseText;
    }
  }
}
```

8

```
xhr.open('GET','someResource');
xhr.send();
```

尽管这里没有使用什么技巧，但却包含了大量的代码，足足有 20 行，还不包括为了方便阅读而添加的空白行。

使用 jQuery 编写的作为就绪处理器主体的等价代码如下所示：

```
$('#someContainer').load('someResource');
```

我非常清楚你喜欢编写和维护哪个代码！下面来仔细考察下这个语句中使用的 jQuery 方法。

8.2.1　使用jQuery加载内容

上一节结尾处的简单 jQuery 语句轻松地从服务器端资源加载内容，它使用了一个最基础，但是很有用的 Ajax 方法：`load()`。这个方法完整的语法描述如下所示。

方法语法：load

load(url, parameters, callback)

向指定的 URL 发起带有可选请求参数的 Ajax 请求。可以指定一个回调函数，并在请求完成并且 DOM 被修改之后调用此函数。响应文本会替换所有匹配元素的内容

参数

url	（字符串）要将请求发送到的服务器端资源的 URL，可以通过选择器（可选）来修改 URL（下面会介绍）
parameters	（字符串|对象|数组）指定要作为请求参数传递的任何数据。这个参数可以是字符串（被用作查询字符串），也可以是对象（其属性被序列化为正确编码的参数，这些参数会被传入请求），或者由对象组成的数组（对象的 name 和 value 属性指定了名称/值对） 如果指定为对象或者数组，则使用 POST 方法来发起请求。如果省略或者指定为字符串，则使用 GET 请求
callback	（函数）一个可选的回调函数，在响应数据被载入匹配集元素后调用。传入此函数的参数是响应文本、状态字符串（通常是 success）以及 XHR 实例 为包装集的每个元素调用一次此函数，并把函数上下文（this）设置为目标元素

返回值

包装集

尽管使用起来很简单，但是这个方法还有一些重要细节需要注意。例如，当使用 parameters 来提供请求参数时，如果使用的是散列对象或者数组，则通过 POST HTTP 方法来发起请求，否

则发起 `GET` 请求。如果想要创建一个带参数的 `GET` 请求，可以将它们作为查询字符串包含在 URL 中。但是这样做的时候要注意，我们有责任确保查询字符串已被正确格式化并且请求参数中的名称和值已进行 URI 编码。使用 JavaScript 的 `encodeURIComponent()`方法很容易做到这一点，或者也可以采用在第 6 章阐述过的 jQuery 的`$.param()`实用函数提供的服务。

大部分时候，我们使用 `load()`方法把整个响应注入包装集的元素中，但是有时我们可能想要筛选作为响应返回的元素。如果要筛选响应元素，jQuery 允许在 URL 上指定选择器，用来限定哪些响应元素会被注入到包装元素中。通过在 URL 中添加紧随空格的选择器后缀来指定选择器。

例如，要筛选响应元素以便只注入`<div>`实例，可以这样写：

```
$('.injectMe').load('/someResource div');
```

当说到提供在请求中提交的数据时，有时候我们会临时拼凑特殊的数据，但是我们经常会发现自己希望获取用户输入到表单控件中的数据。

正如你可能期望的那样，jQuery 为此提供了一些帮助。

序列化表单数据

如果作为请求参数来发送的数据来自于表单控件，则用来帮助创建查询字符串的 jQuery 方法是 `serialize()`，其语法如下所示。

方法语法：serialize

serialize()

根据包装集中所有成功的表单元素或者包装集的表单中所有成功的表单元素，创建正确格式化和编码的查询字符串

参数

无

返回值

格式化的查询字符串

`serialize()`方法非常智能，它只从包装集中的表单控件元素上以及被认为是成功的有资格的元素上收集信息。根据 HTML 规范[①]的规则，成功的控件是指作为表单提交的一部分而被包含的控件。未选中的复选框和单选按钮、未选中的下拉列表以及禁用的控件都被认为是不成功的控件，因此不会参与表单提交，这些控件也会被 `serialize()`忽略。

如果想要以 JavaScript 数组的形式（而不是查询字符串）来获取表单数据，可以使用 jQuery 提供的 `serializeArray()`方法。

① HTML 4.01 规范 17.13.2 节，“Successful controls”：http://www.w3.org/TR/html401/interact/forms.html#h-17.13.2。

方法语法：serializeArray

serializeArray()

把所有成功的表单控件的值收集到包含控件名称和值的对象数组中

参数

无

返回值

由表单数据组成的数组

serializeArray()返回的数组由匿名的对象实例组成，每个实例包含一个 name 属性和一个 value 属性，它们分别包含了每个成功表单控件的名称和值。这是（不是偶然地）适合用来传递给 load()方法来指定请求参数数据的格式之一。

有了 load()方法可供利用，下面就来解决一些很多 Web 开发人员遇到的现实世界中的常见问题。

8.2.2　加载动态的HTML片段

在商业应用中（特别是对于贸易网站），我们经常想要从服务器获取实时数据以便向用户展示最新的信息。毕竟，我们不想使消费者误以为他们可以购买一些不可能买到的东西，对吧？在本节中，我们将着手开发一个贯穿本章教程的页面。这个页面是一个名为 The Boot Closet 的虚构公司（摩托靴进销存在线零售商）网站的一部分。与其他具有固定产品目录的在线零售商不同，这个网站的进销存清单是变化的，取决于经营者当天完成了什么交易和已经从库存里卖掉了哪些东西。对于我们来说确保始终显示最新的信息是至关重要的。

开始创建页面（忽略站点导航和其他样板以便专注于眼前的课程），我们希望为客户呈现一个包含当前可用款式的下拉列表，当一种款式被选择后就显示这个款式相关的详细信息。初始显示时，页面看起来将会如图 8-2 所示。



图 8-2　贸易页面的初始显示，用一个简单的下拉列表来吸引用户的单击

在页面第一次加载后，会显示出带有当前库存中可用款式清单的下拉列表。当没有选中任何款式时，就显示一个帮助信息—choose a style—作为选择项的占位符。这可以吸引用户与下拉列表进行交互，当用户从下拉列表选中一种靴子款式时，我们需要做以下事情：

- 在下拉列表下面的区域显示有关这个款式的详细信息；
- 删除—choose a style—项。如果用户选中了一个款式，这个选择项就完成了自己的使命而不再有意义了。

下面从分析用来定义这个页面结构主体的 HTML 标记开始：

```
<body>
  <div id="banner">
    <img src="images/banner.boot.closet.png" alt="The Boot Closet"/>
  </div>
  <div id="pageContent">
    <h1>Choose your boots</h1>
    <div>
      <div id="selectionsPane">                                    ❶ 包含下拉列表控件
        <label for="bootChooserControl">Boot style:</label> 
        <select id="bootChooserControl" name="bootStyle"></select>
      </div>
      <div id="productDetailPane"></div>                           ❷ 为产品的详细信息预留空间
    </div>
  </div>
</body>
```

这段标记的内容不多，是吧？

遵循不唐突 JavaScript 的规则，我们已经在外部样式表中定义了所有的视觉渲染信息，并且没有在 HTML 标记中包含任何行为方面的代码。

这段标记最有趣的部分是一个用来容纳允许客户选择靴子款式的`<select>`元素的容器❶，以及另一个用来注入产品详细信息的容器❷。

注意，在用户与页面交互之前，需要向靴子款式控件添加其选项元素。下面就来着手为该页面添加必要的行为。

首先添加的是一个用来获取和填充靴子款式下拉列表的 Ajax 请求。

注意　在大部分情况下，类似这样的初始值应该在发送 HTML 到浏览器之前在服务器上处理。但是在有的情况下通过 Ajax 预先获取数据也可能是合适的，我们这么做仅仅是为了教学目的。

为了向靴子款式控件添加选项，我们定义了一个就绪处理器并在其中使用便捷的 `load()` 方法：

```
$('#bootChooserControl').load('/jqia2/action/fetchBootStyleOptions');
```

是不是很简单？这个语句唯一复杂的地方是 URL（并非真的很长或者很复杂），它指定了到服务器端名为 `fetchBootStyleOptions` 活动的请求。

使用 Ajax 的一个好处（在 jQuery 的帮助下这个好处更加明显）是，它完全不依赖于服务器端的技术。很明显服务器端所选择的技术对 URL 的结构有影响，但是除此之外无需太担心服务器发生的事情。我们只需简单地发起 HTTP 请求，有时带有合适的参数数据，并且只要服务器返回预期的响应，我们就不必关心服务器是由 Java、Ruby、PHP，还是老式的 CGI 来驱动的。

在这种特定情况下，我们期望服务器端资源返回表示靴子款式选项的 HTML 标记——可能是来自于库存数据库。我们伪造的后端代码返回的响应如下所示：

```
<option value="">&mdash; choose a style &mdash;</option>
<option value="7177382">Caterpillar Tradesman Work Boot</option>
<option value="7269643">Caterpillar Logger Boot</option>
<option value="7332058">Chippewa 9" Briar Waterproof Bison Boot</option>
<option value="7141832">Chippewa 17" Engineer Boot</option>
<option value="7141833">Chippewa 17" Snakeproof Boot</option>
<option value="7173656">Chippewa 11" Engineer Boot</option>
<option value="7141922">Chippewa Harness Boot</option>
<option value="7141730">Danner Foreman Pro Work Boot</option>
<option value="7257914">Danner Grouse GTX Boot</option>
```

这个响应随后被嵌入到`<select>`元素中，生成一个具有完全功能的控件。

接下来的操作是为下拉列表添加行为这样它可以对改变做出反应，从而实现前面列出的职责。为此编写的代码稍微有点复杂：

```
$('#bootChooserControl').change(function(event){      ◁ ❶ 建立改变处理器
  $('#productDetailPane').load(                          ❷ 获取和显示产品详细信息
    '/jqia2/action/fetchProductDetails',
    {style: $(event.target).val()},
    function() { $('[value=""]',event.target).remove(); }
  );
});
```

在这段代码中，我们选取靴子款式的下拉列表并为其绑定一个 `change` 处理器❶。在改变处理器的回调中（每当客户改变选择项都会调用此回调），我们通过向事件目标（也就是触发事件的`<select>`元素）应用 `val()`方法来获取选择项的当前值。然后对 `productDetailPane` 元素再次使用 `load()`方法❷来向服务器端资源（即 `fetchProductDetails`）发起 Ajax 回调，并传入参数名为 `style` 的款式值。

在客户选择了一个可用的靴子款式后，页面将会如图 8-3 所示。

在就绪处理器中执行的最值得关注的操作是 `load()`方法的使用，用来快速轻松地从服务器获取 HTML 片段并把它们作为现有元素的子元素而插入到 DOM 中。这个方法非常方便并且适合由服务器（具有服务器端模板功能例如 JSP 和 PHP 技术）支持的 Web 应用程序。

代码清单 8-4 显示了 The Boot Closet 页面的完整代码，可以从 http://localhost[:8080]/jqia2/chapter8/bootcloset/phase.1.html 找到。在本章的学习过程中，我们会再次访问这个页面为其添加更多的功能。

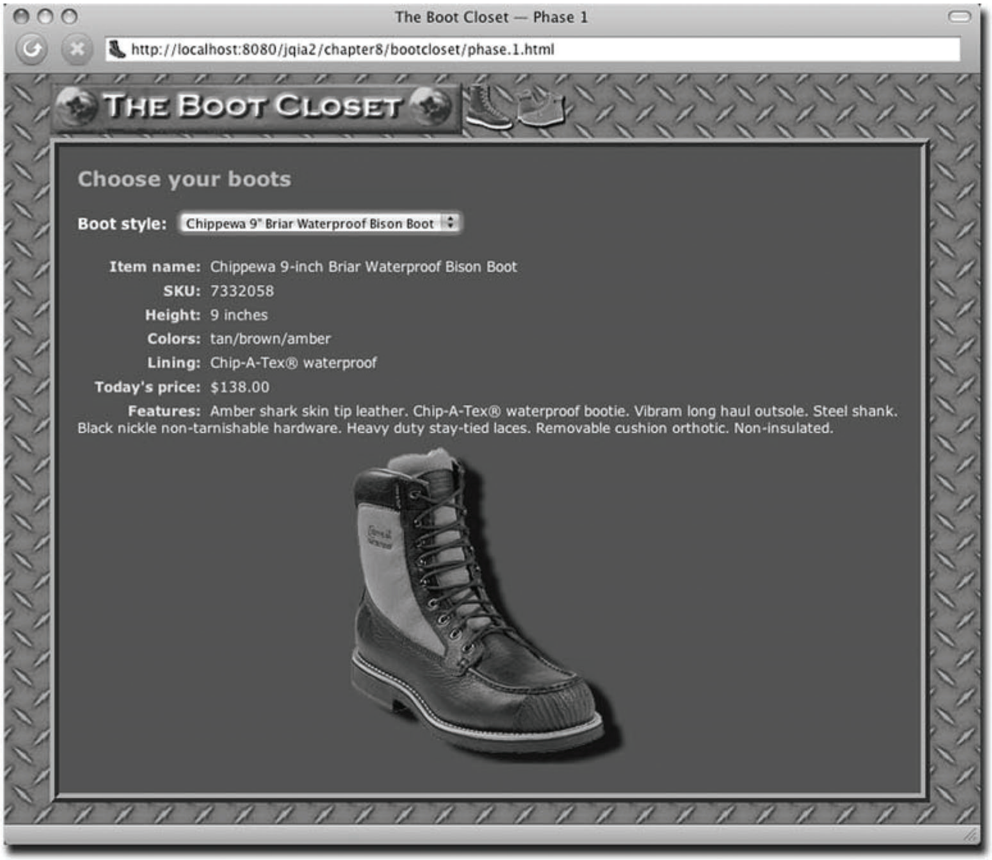


图 8-3　服务器端资源返回预格式化的 HTML 片段来显示靴子的详细信息

代码清单 8-4　The Boot Closet 贸易页面的第一阶段代码


```
<!DOCTYPE html>
<html>
  <head>
    <title>The Boot Closet &mdash; Phase 1</title>
    <link rel="stylesheet" type="text/css" href="../../styles/core.css">
    <link rel="stylesheet" type="text/css" href="bootcloset.css">
    <link rel="icon" type="image/gif" href="images/favicon.gif">
    <script type="text/javascript"
            src="../../scripts/jquery-1.4.js"></script>
    <script type="text/javascript"
            src="../../scripts/jqia2.support.js"></script>
    <script type="text/javascript">
      $(function() {

        $('#bootChooserControl')
          .load('/jqia2/action/fetchBootStyleOptions');
```

```
        $('#bootChooserControl').change(function(event){
          $('#productDetailPane').load(
            '/jqia2/action/fetchProductDetails',
            {style: $(event.target).val()},
            function() { $('[value=""]',event.target).remove(); }
          );
        });

      });
    </script>
  </head>

  <body>

    <div id="banner">
      <img src="images/banner.boot.closet.png" alt="The Boot Closet"/>
    </div>

    <div id="pageContent">

      <h1>Choose your boots</h1>

      <div>

        <div id="selectionsPane">
          <label for="bootChooserControl">Boot style:</label> 
          <select id="bootChooserControl" name="bootStyle"></select>
        </div>

        <div id="productDetailPane"></div>

      </div>

    </div>

  </body>

</html>
```

如果需要获取 HTML 片段来填充一个元素（或者一组元素）的内容，`load()`方法就非常有用。但是有时候我们希望对如何创建 Ajax 请求进行更多的控制，或者需要对响应主体中的返回数据进行更加复杂的操作。

下面继续考察 jQuery 为这些复杂的情况提供了哪些功能。

8.3　发起 GET 和 POST 请求

`load()`方法可以发起 `GET` 或者 `POST` 请求，这取决于如何提供请求参数数据（如果有的话），但是有时候我们想对要使用哪个 HTTP 方法进行更多的控制。为什么我们要关心使用哪个 HTTP 方法呢？因为服务器关心。

传统上，Web 作者使用 `GET` 和 `POST` 方法非常随意，不关心 HTTP 协议规定如何使用这些方法。这两个方法的使用规则如下所示。

- `GET` 请求——意图是幂等的：相同的 `GET` 操作无论进行一次、两次还是三次操作，返回的结果应该是完全相同的（假设没有其他的力量在起作用来改变服务器状态）。
- `POST` 请求——可以是非幂等的：发送到服务器的数据可以用来改变应用程序的模型状态。例如，添加或者更新数据库中的记录或者从服务器删除信息。

因此，GET 请求应该只用于获取数据时使用，正如它的名称所暗示的那样。可能需要为 GET 请求向服务器发送一些参数数据。例如，为了获取颜色信息而发送款式号码。但是如果向服务器发送数据的目的是为了进行改变，就应该使用 POST 请求。

警告 这是理论之外的内容。浏览器会根据使用的 HTTP 方法来决定是否缓存，GET 请求非常容易被缓存。请使用正确的 HTTP 方法以确保不会违背浏览器或者服务器对请求意图的期望。这只是对 REST 式理论领域的一个初探，在 REST 式理论中也可以使用其他的 HTTP 方法比如 PUT 和 DELETE。但是为了达到目的，我们这里只讨论 GET 和 POST 方法。

考虑到这一点，如果回顾一下 The Boot Closet 第一阶段的实现（代码清单 8-4），就会发现我们做错了！因为如果我们向参数数据提供一个散列对象，jQuery 就会发出 POST 请求，真正应该发出 GET 请求时却发出了 POST 请求。在 Firefox 下显示页面时看一下 Firebug 的记录（如图 8-4 所示）就可以看到第二个请求（当我们从款式下拉列表中选择一项时被提交的请求）的确是 POST。

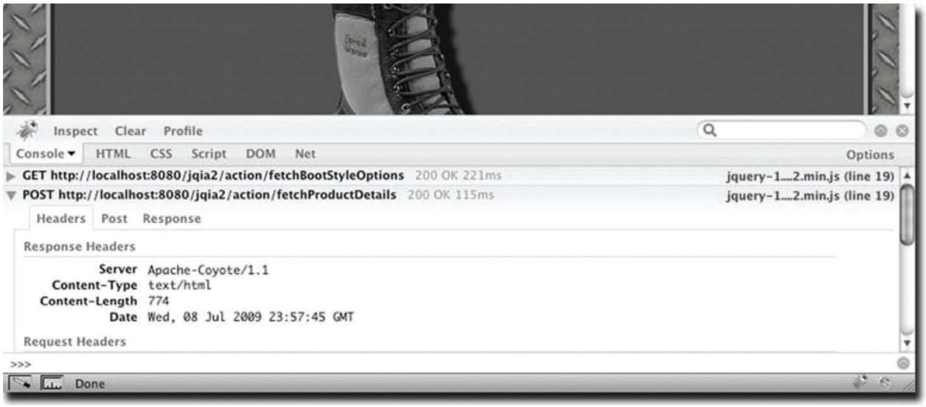


图 8-4　查看 Firefox 控制台就会发现应该发出 GET 请求时发出的却是 POST 请求

是否真的很重要？这取决于你自己，但是如果想按照 HTTP 期望的方式来使用它，那么获取靴子详细信息的请求应该是 GET 而不是 POST。

我们可以简单地创建指定请求信息的参数为字符串而不是散列对象（稍后会再次探讨这个情况），但是此刻先来利用 jQuery 提供的另一种方式来发起 Ajax 请求。

获取 Firebug

开发一个 DOM 脚本应用程序而不借助调试工具，这就像带着电焊手套在音乐会中弹钢琴。这不是自找苦吃吗？

你的开发工具箱中要有一个重要的工具——Firebug，它是 Firefox 浏览器的一个插件。如图 8-4 所示，在页面开发的过程中 Firebug 不仅允许我们查看 JavaScript 控制台，还可以查看动态的 DOM、CSS、脚本以及页面上其他方面的内容。

> 和当前的目标最为相关的一个特征是，Firebug 拥有将 Ajax 请求连同其请求和响应信息一起记录下来的能力。
>
> 对于非 Firefox 的浏览器，可以使用 Firebug Lite 版本，它会在调试的时候作为一个 JavaScript 库来加载。
>
> 你可以从 http://getfirebug.com 下载 Firebug，在http://getfirebug.com/lite.html下载 Firebug Lite。
>
> Google 的 Chrome 浏览器内置了类似于 Firebug 的调试能力，可以通过打开它的 Developer Tools 来显示调试器（看下菜单来查找此项——它经常会改变位置）。

8.3.1　使用GET获取数据

jQuery 提供了发起 `GET` 请求的几种方式，与 `load()`不同，这些方式不是作为应用到包装集上的 jQuery 方法来实现的，而是提供了几个实用函数来发起各种不同类型的 `GET` 请求。第 1 章里曾提到过，jQuery 实用函数是位于 `jQuery` 全局名称（及其别名`$`）命名空间下面的顶级函数。

如果我们想要从服务器获取一些数据，然后决定如何处理这些数据（而不是让 `load()`方法把这些数据设置为 HTML 元素的内容），就可以使用`$.get()`实用函数。它的语法如下所示。

函数语法：`$.get`

`$.get(url, parameters, callback, type)`

使用指定的 URL，用任何已传入的参数作为查询字符串向服务器发起 `GET` 请求

参数

`url`	（字符串）通过 `GET` 方法接触的服务器端资源的 URL
`parameters`	（字符串|对象|数组）指定将要作为请求参数传递的任何数据。这个参数可以是字符串（被用作查询字符串）、对象（其属性被序列化为正确编码的参数，这些参数会被传入请求）或者由对象组成的数组（对象的 `name` 和 `value` 属性指定了名称/值对）
`callback`	（函数）一个可选的回调函数，在请求成功完成时调用。传入回调的第一个参数是响应主体（根据设置的 `type` 参数来解析响应主体），第二个参数是文本信息。第三个参数包含对 XHR 实例的引用
`type`	（字符串）指定如何解析响应主体，（可选）可以是下列类型中的一种：`html`、`text`、`xml`、`json`、`script` 或者 `jsonp`。查看本章随后对`$.ajax()`的描述以了解更多的细节

返回值

XHR 实例

$.get()实用函数允许使用多种方式来发起 GET 请求。可以使用多种方便的格式来指定请求参数（如果有的话），提供响应成功时所调用的回调，甚至控制响应主体该如何被解析以及传入回调。如果这还是不够通用的话，我们稍后将看到一个更加通用的函数——$.ajax()。

当考察$.ajax()实用函数时，我们会更加详细地审查 type 参数，不过目前先依据响应的内容类型让它默认为 html 或 xml[①]。

在 The Boot Closet 页面使用$.get()函数替换 load()方法，如代码清单 8-5 所示。

代码清单 8-5 改变 The Boot Closet 来使用 GET 获取款式信息

```
$('#bootChooserControl').change(function(event){
  $.get(                                                    ❶ 发起 GET 请求
    '/jqia2/action/fetchProductDetails',
    {style: $(event.target).val()},
    function(response) {
      $('#productDetailPane').html(response);               ❷ 插入响应 HTML
      $('[value=""]',event.target).remove();
    }
  );
});
```

页面在第二阶段的改变很小，但是很重要。我们调用$.get()❶来取代 load()，传入相同的 URL 和相同的请求参数。因为$.get()不会自动把响应插入到 DOM 中的任何地方，我们需要自己来做这件事情，可以通过调用 html()方法来完成❷。

这个版本的页面代码可以从 http://localhost[:8080]/jqia2/chapter8/bootcloset/phase.2.html 找到，当页面显示出来后，我们从下拉列表中选择一个款式，就会看到一个 GET 请求被发送出来，如图 8-5 所示。

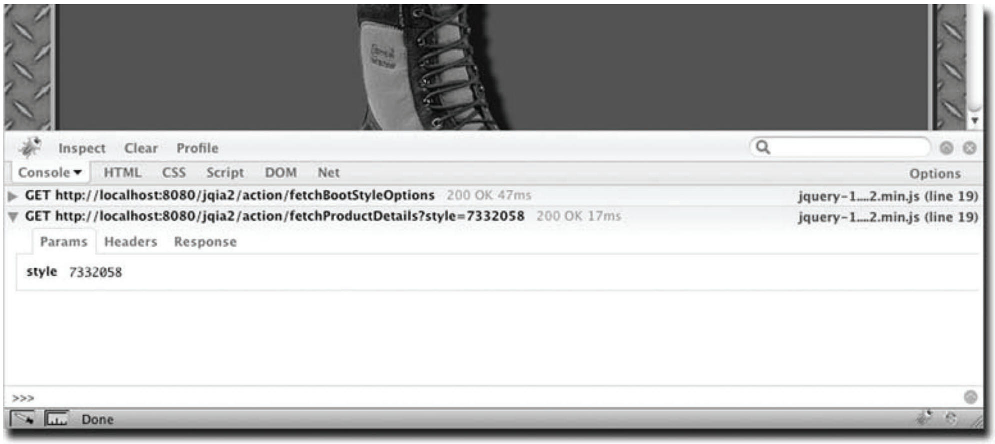


图 8-5 现在可以看到的第二个请求是 GET 而不是 POST，与操作相对应

① type 参数的默认值会根据响应头中定义的内容类型来智能确定，比如 Content-Type:text/html 表示响应主体为 HTML 片段，Content-Type:application/json 表示响应主体为 JSON 格式的字符串。

本例中，我们从服务器返回格式化的HTML并把它插入到DOM中，但是我们可以从$.get()的 type 参数看到，除了 HTML 之外还可以使用其他很多类型。事实上，术语 Ajax 是作为首字母缩写 AJAX 被人们熟知的，其中 X 代表 XML。

当向 type 传入 xml（记住我们稍后会详细讨论 type 参数）并且从服务器返回 XML 时，传入回调的数据是解析后的 XML DOM。XML 的灵活性非常适合处理本质上高度分层的数据，但是遍历和获取 XML 的数据却是痛苦的。下面来看另一个 jQuery 实用函数，它在数据需求较为简单的时候非常有用。

8.3.2　获取JSON数据

在前一节我们谈到，当 XML 文档从服务器返回时，它会被自动解析并且使结果 DOM 可供回调函数使用。当 XML 显得大材小用或者不适合作为数据传输机制时，一般会使用 JSON 来代替它。这样选择的一个原因是 JSON 非常容易融入客户端脚本。jQuery 可以使这个操作更加容易。

如果我们知道响应主体将会是 JSON 的时候，调用$.getJSON()实用函数可以自动地解析返回的 JSON 字符串，使结果 JavaScript 数据项可供该函数的回调函数使用。这个实用函数的语法如下所示。

函数语法：$.getJSON

$.getJSON(url, parameters, callback)

使用指定的 URL 和作为查询字符串的任何传入的参数来向服务器发起 GET 请求。把响应解析为 JSON 字符串，并且把结果数据传递给回调函数

参数

url	（字符串）通过 GET 方法接触的服务器端资源的 URL
parameters	（字符串|对象|数组）指定将要作为请求参数传递的任何数据。这个参数可以是字符串（被用作查询字符串）、对象（其属性被序列化为正确编码的参数，这些参数会被传入请求）或者由对象组成的数组（对象的 name 和 value 属性指定了名称/值对）
callback	（函数）在请求完成的时候调用的函数。传入这个回调的第一个参数是把响应主体作为 JSON 表示法解析所得到的数据值，第二个参数是状态文本。第三个参数提供了一个对 XHR 实例的引用

返回值

XHR 实例

有时我们想从服务器获取数据而无需付出处理 XML 的开销，那这个函数（只是个 type 为 json 的$.get()的便捷函数）用起来会非常棒。

当需要发起 GET 请求时，jQuery 为我们提供了强大的工具$.get()和$.getJSON()，但是仅仅依靠 GET 是不行的！

8.3.3 发起POST请求

"有时你想要坚果，有时却不想要。"[①]在 Almond Joy 和 Mounds 这两种糖果之间做出选择和选择向服务器发起哪种请求的心情是一样的。有时我们想发起 GET 请求，但是其他时候我们想（或者甚至是需要）发起 POST 请求。

有很多理由可以说明为什么我们可能选择 POST 而不是 GET。首先 HTTP 协议的规定是，POST 用于任何非幂等的请求。因此，如果请求有可能导致服务器端状态的改变（结果是响应改变了），那就应该使用 POST 请求。此外，从公认的惯例和约定来看，当要向服务器传递的数据超出了查询字符串中可以通过 URL 传递的数量——这个限定数量因浏览器而异，此时就必须使用 POST 操作。有时候，我们接触的服务器端资源可能只支持 POST 操作，或者甚至根据请求所用的方法是 GET 还是 POST 来执行不同功能。

对于那些要求或者强制使用 POST 的情况，jQuery 提供了$.post()实用函数除了采用 POST HTTP 方法，这个函数的运行方式类似于$.get()。它的语法如下所示。

函数语法：$.post

$.post(url, parameters, callback, type)

使用指定的 URL 和作为请求主体的任何传入的参数来向服务器发起 POST 请求

参数

url	（字符串）通过 POST 方法接触的服务器端资源的 URL
parameters	（字符串|对象|数组）指定将要作为请求参数传递的任何数据。这个参数可以是字符串（被用作查询字符串）、对象（其属性被序列化为正确编码的参数，这些参数会被传入请求）或者由对象组成的数组（对象的 name 和 value 属性指定了名称/值对）
callback	（函数）当请求完成时调用的函数。传入回调的第一个参数是响应主体，第二个参数是状态文本。第三个参数提供了对 XHR 实例的引用
type	（字符串）（可选地）指定如何解析响应主体，可以是这些类型中的一种：html、text、xml、json、script 或者 jsonp。查看$.ajax()的描述以了解更多的细节

返回值

XHR 实例

除了发起 POST 请求，使用$.post()和使用$.get()是一样的。jQuery 会负责在请求主体（而不是查询字符串）中传递请求数据的细节，并且设置合适的 HTTP 方法。

① 这是 1970 年 Peter Paul 公司在宣传 Almond Joy 和 Mounds 两种糖果时的广告语，后来该公司被好时公司收购。

8

现在回到 The Boot Closet 项目，我们已经有了一个良好的开端，但是在购买一双靴子时，只能选择款式是远远不够的；客户肯定希望选择想要的颜色，而且当然也需要指定靴子尺寸。我们将使用这些额外的需求来展示如何解决在线 Ajax 论坛中经常被问到的问题之一，就是……

8.3.4　实现级联下拉列表

级联下拉列表的实现（随后的下拉列表选项取决于之前在下拉列表中的选择）已经成为 Ajax 在网络上的典型应用。虽然你会发现数以千计，也许是数以万计的解决方案，但是将在 The Boot Closet 页面上实现的解决方案将会证明，用 jQuery 来实现这个过程是多么地简单。

我们曾介绍过，使用服务器提供的选项数据来动态地加载下拉列表是非常方便的。下面将看到通过级联关系把多个下拉列表绑定在一起的方法，这也很方便。

下面列出下一阶段需要对页面做出的改变。

- 添加颜色和尺寸的下拉列表。
- 当选中一个款式时，为颜色下拉列表添加可供当前款式使用的颜色选项。
- 当选中一个颜色时，为尺寸下拉列表添加可供当前选中的款式和颜色组合使用的尺寸选项。
- 确保三个下拉列表的一致性。这包括新创建的下拉列表被使用过一次后，要从中删除--please make a selection--选项，而且还要确保三个下拉列表绝不显示一个无效的组合。

我们打算再次使用 `load()`，这次强制它发起 `GET` 请求而不是 `POST` 请求。这并不意味着我们有任何抵制`$.get()`的想法，只是当使用 Ajax 加载 HTML 片段时，`load()`看起来更加自然。

首先，来看看用来定义另外几个下拉列表的新的 HTML 标记。为选择的元素定义的新容器包含三个带有标签的元素：

```
<div id="selectionsPane">
  <label for="bootChooserControl">Boot style:</label>
  <select id="bootChooserControl" name="style"></select>
  <label for="colorChooserControl">Color:</label>
  <select id="colorChooserControl" name="color" disabled="disabled"></select>
  <label for="sizeChooserControl">Size:</label>
  <select id="sizeChooserControl" name="size" disabled="disabled"></select>
</div>
```

之前的款式选择元素依然存在，并且增加了另外两个元素：一个是颜色选择元素，另一个是尺寸选择元素，每一个都被初始化为空并且处于禁用状态。

这很容易，这段代码负责在页面中添加几个新标记。下面来添加它们的行为。

款式选择下拉列表现在必须履行双重职责。当从下拉列表中选择一项时，它不仅需要继续获取并显示靴子的详细信息，而且其改变处理器现在还必须填充并启用颜色选择下拉列表，填充的颜色取决于选中的款式。

首先来重构一下获取详细信息的代码。我们希望使用 `load()`，但是也希望强制发起 `GET` 请求而不是之前发起的 `POST` 请求。为了让 `load()`使用 `GET`，我们需要传递一个字符串而不是散列对象，来指定请求参数数据。幸运的是，在 jQuery 的帮助下我们无需自己创建这个字符串。

修改款式下拉列表的改变处理器，其第一部分如下所示：

```
$('#bootChooserControl').change(function(event){
  $('#productDetailPane').load(
    '/jqia2/action/fetchProductDetails',
    $(this).serialize()                          <-- 提供查询字符串
  );
  // 更多的代码
});
```

通过使用 serialize()方法，我们创建了款式下拉列表值的字符串表示，从而强制 load()方法发起 GET 请求，以满足需要。

改变处理器需要执行的第二个职责是，为选中的款式加载带有合适值的颜色选择下拉列表，然后启用它。下面来看看要添加到处理器的其余代码：

```
$('#colorChooserControl').load(
  '/jqia2/action/fetchColorOptions',
  $(this).serialize(),                           ❶ 获取并加载颜色选项
  function(){
    $(this).attr('disabled',false);         <--
    $('#sizeChooserControl')         ❸ 禁用和清空尺   ❷ 启用颜色控件
      .attr('disabled',true)            寸控件
      .html("");
  }
);
```

这段代码应该看起来很熟悉。这只是 load()的另一种用法，这次引用了一个名为 fetchColorOptions 的动作，目的是返回一组格式化的<option>元素来呈现可供选中款式（再次被作为请求数据传递）使用的颜色❶。这一次，我们还指定了一个要在 GET 请求成功返回响应时执行的回调函数。

在这个回调函数中，我们执行了两个重要的任务。首先，启用颜色选择控件❷。调用 load()会插入<option>元素，但是如果填充之后不启用颜色控件的话它仍然将处于禁用状态。

其次，回调禁用并且清空了尺寸选择控件❸，但这是为什么呢？（停顿片刻来好好地想一想。）

尺寸控件在款式选择器的值第一次改变时就已经处于禁用和清空状态，那后来会怎么样呢？如果客户在选择款式和颜色后（我们将会很快看到尺寸控件的填充结果），他改变了选中的款式，会怎么样？因为显示的尺寸取决于款式和颜色的组合，所以之前显示的尺寸已经不再合适，也不能为已选择项呈现一致的界面。因此，每当款式改变时，我们都需要清空尺寸选项并且重置尺寸控件到其初始状态。

在坐下来享受一杯可口的饮料之前，还有很多的工作要做。我们仍然需要为颜色选择下拉列表添加行为，以便使用选中的款式和颜色值来获取并加载尺寸选择下拉列表。完成这项工作的代码遵循那个我们早已熟悉的模式：

```
$('#colorChooserControl').change(function(event){
  $('#sizeChooserControl').load(
    '/jqia2/action/fetchSizeOptions',
    $('#bootChooserControl,#colorChooserControl').serialize(),
```

```
      function(){
        $(this).attr('disabled',false);
      }
    );
  });
```

在一个改变事件中，尺寸信息是通过 fetchSizeOptions 动作来获取的（传入靴子款式和选中的颜色），然后启用尺寸控件。

还有一件事情需要完成。当最初填充每个下拉列表时，会默认显示拥有空白值和显示文本为--choose a something--的<option>元素。你可能会想起在此页面的前几个阶段中，我们添加代码，以便在从款式下拉列表中选择一项时删除这个选项。

好的，我们可以为款式和颜色下拉列表添加这样的代码到改变处理器，并且为尺寸下拉列表添加这样的行为（目前还没有为尺寸下拉列表绑定行为）。让我们来优雅地完成这个目的。

事件模型有一个能力——事件冒泡，但许多 Web 开发人员往往忽略它。页面开发者通常只关注事件的目标，而忘记了事件会沿着 DOM 树向上冒泡，处理器可以使用更加通用的方式处理事件，而不是在目标元素的级别上处理它们。

其实可以用完全相同的方式从这三个下拉列表中的任意一个删除拥有空白值的选项，而不管哪个下拉列表才是事件的目标。方法是在 DOM 的更高层次创建用来识别和处理改变事件的单个处理器，从而避免在三个地方重复相同的代码。

回忆一下文档结构，三个下拉列表都被包含在 id 为 selectionsPane 的<div>元素中。我们可以使用一个监听器为所有三个下拉列表删除临时选项：

```
$('#selectionsPane').change(function(event){
  $('[value=""]',event.target).remove();
});
```

任何一个内部的下拉列表发生改变事件时都会触发这个监听器，而且会删除事件目标（发生改变的下拉列表）上下文中的拥有空白值的选项。这是不是很灵活？

使用事件冒泡在较低级别的处理器中避免重复相同的代码，可以真正提升你编写脚本的水平！

这样我们就完成了 The Boot Closet 第三阶段的代码，在页面中添加了级联下拉列表，如图 8-6 所示。如果页面上后一个下拉列表中的值取决于前一个下拉列表的选择，我们就可以使用这种技术。这个阶段的页面可以从 URL http://localhost[:8080]/jqia2/chapter8/bootcloset/phase.3.html 中找到。



图 8-6　The Boot Closet 第三阶段展示了实现级联下拉列表是多么地容易

这个页面的完整代码如代码清单 8-6 所示。

代码清单 8-6　The Boot Closet，现在拥有级联下拉列表了

```
<!DOCTYPE html>
<html>
  <head>
    <title>The Boot Closet &mdash; Phase 3</title>
    <link rel="stylesheet" type="text/css" href="../../styles/core.css">
    <link rel="stylesheet" type="text/css" href="bootcloset.css">
    <link rel="icon" type="image/gif" href="images/favicon.gif">
    <script type="text/javascript"
            src="../../scripts/jquery-1.4.js"></script>

    <script type="text/javascript"
            src="../../scripts/jqia2.support.js"></script>
    <script type="text/javascript">
      $(function() {

        $('#bootChooserControl')
          .load('/jqia2/action/fetchBootStyleOptions');
```

```
      $('#bootChooserControl').change(function(event){
        $('#productDetailPane').load(
          '/jqia2/action/fetchProductDetails',
          $(this).serialize()
        );
        $('#colorChooserControl').load(
          '/jqia2/action/fetchColorOptions',
          $(this).serialize(),
          function(){
            $(this).attr('disabled',false);
            $('#sizeChooserControl')
              .attr('disabled',true)
              .html("");
          }
        );
      });

      $('#colorChooserControl').change(function(event){
        $('#sizeChooserControl').load(
          '/jqia2/action/fetchSizeOptions',
          $('#bootChooserControl,#colorChooserControl').serialize(),
          function(){
            $(this).attr('disabled',false);
          }
        );
      });

      $('#selectionsPane').change(function(event){
        $('[value=""]',event.target).remove();
      });

    });
  </script>
</head>

<body>

  <div id="banner">
    <img src="images/banner.boot.closet.png" alt="The Boot Closet"/>
  </div>

  <div id="pageContent">

    <h1>Choose your boots</h1>

    <div>

      <div id="selectionsPane">
        <label for="bootChooserControl">Boot style:</label>
        <select id="bootChooserControl" name="style"></select>
        <label for="colorChooserControl">Color:</label>
        <select id="colorChooserControl" name="color"
                disabled="disabled"></select>
        <label for="sizeChooserControl">Size:</label>
        <select id="sizeChooserControl" name="size"
                disabled="disabled"></select>
      </div>
```

```
        <div id="productDetailPane"></div>
        </div>

      </div>

    </body>

  </html>
```

如上所示，有了 load()方法和各种 GET 和 POST 的 jQuery Ajax 函数，我们就可以对如何发起请求以及如何在请求完成时公布通知进行一些控制。但是有时我们需要对 Ajax 请求进行完全控制，jQuery 有一个方法可以实现这个目的。

8.4 完全控制 Ajax 请求

目前为止，我们介绍了适用于各种情况的函数和方法，但是有时我们想自己控制所有的细节。本节将探索 jQuery 如何让我们运用这种控制权。

8.4.1 发起带所有参数的Ajax请求

如果我们想要或者需要控制 Ajax 请求的各种细节，可以利用 jQuery 提供的名为$.ajax()的通用实用函数来发起 Ajax 请求。实际上，所有其他用来发起 Ajax 请求的 jQuery 功能最终都是使用这个函数来发起请求。它的语法如下所示。

函数语法：$.ajax

$.ajax(options)

使用传入的选项来发起 Ajax 请求，以便控制如何创建请求以及如何通知回调

参数

options （对象）一个对象，其属性定义了这个操作的参数。详情参见表 8-2

返回值

XHR 实例

8

看起来很简单，不是吗？但是不要被表象欺骗了。options 参数可以指定的值非常多，它们可以调整这个函数的操作。这些选项（按照重要性和使用频率排序）的定义参见表 8-2。

表8-2 **$.ajax()**实用函数的选项

名 称	描 述
url	（字符串）请求的URL
type	（字符串）要使用的HTTP方法。通常是POST或者GET。如果省略则默认为GET
data	（字符串|对象|数组）定义将要作为查询参数传入请求的值。如果请求是GET，则把数据作为查询字符串传递。如果请求是POST，则把数据作为请求主体传递。无论哪种情况，$.ajax()实用函数都会负责对值进行编码。这个参数可以是字符串（被用作查询字符串）、对象（其属性被序列化）或者由对象组成的数组（对象的name和value属性指定了名称/值对）

（续）

名　　称	描　　述
dataType	（字符串）一个关键字，用来识别预计由响应返回的数据类型。这个参数决定在向回调函数传递数据之前对数据的后期处理（如果有的话）。有效值如下所示 ❑ xml——将响应文本作为XML文档解析并将结果XML DOM传入回调函数 ❑ html——将未经处理的响应文本传入回调函数。对返回的HTML片段中的任何<script>块求值 ❑ json——将响应文本作为JSON字符串来求值，并且将结果对象传入回调 ❑ jsonp——除了允许远程脚本之外其他都类似于json，假设远程服务器支持jsonp ❑ script——将响应文本传入回调函数。在调用回调函数之前，将响应作为一个（或多个）JavaScript语句来处理 ❑ text——假定响应文本是普通文本 服务器资源负责设置合适的content-type响应头 如果省略这个属性，则会将响应文本传入回调函数而不做任何处理或求值
cache	（布尔）如果为false，则确保浏览器不会缓存响应。默认为true，除非指定dataType为script或jsonp
context	（元素）指定某个元素为与这个请求相关的所有回调函数的上下文
timeout	（数值）为Ajax请求设置以毫秒为单位的超时时间。如果请求不能在超时时间到期之前完成，则会终止请求并调用处理错误的回调函数（如果有定义）
global	（布尔）如果为false，则禁止触发全局Ajax事件。这个事件是jQuery特定的自定义事件，它们会在处理Ajax请求进程中的各个时间点或条件下触发。我们会在8.4.3节中详细讨论这些事件。如果省略，则默认值（true）表示要启用全局事件触发
contentType	（字符串）指定请求内容的类型。如果省略，则默认值为application/x-www-form-urlencoded，与表单提交所使用的默认值的类型相同
success	（函数）如果对请求的响应显示的是成功状态码则调用此函数。将响应主体作为这个函数的第一个参数返回，并且根据dataType属性的规范来对其进行求值。第二个参数是包含状态值的字符串——在这种情况下，总是success。第三个参数提供了一个对XHR实例的引用
error	（函数）如果对请求的响应返回的是错误状态码则调用此函数。传入这个函数的参数有三个：XHR实例、状态消息字符串（在这种情况下，是error、timeout、notmodified或者parseerror之一）以及可选的异常对象（有时从XHR实例返回，如果有的话）
complete	（函数）在请求结束时调用的函数。传入的参数有两个：XHR实例和状态信息字符串（success或error）。如果也指定了success或error回调函数，则会在调用回调函数之后调用该函数
beforeSend	（函数）在发起请求之间调用的函数。XHR实例被传入这个函数，它可以用来设置自定义头或者执行其他预请求操作。从这个处理器返回false将会取消请求
async	（布尔）如果指定为false，则将请求作为同步请求来提交。默认请求是异步的
processData	（布尔）如果设置为false，则阻止将传入的数据处理为URL编码的格式。默认情况下，数据被处理为URL编码格式，该格式适合与请求类型application/x-www-form-urlencoded一起使用
dataFilter	（函数）用来筛选响应数据的回调函数。向这个函数传入原始的响应数据和dataType值，这个函数会返回"净化"后的数据
ifModified	（布尔）如果设置为true，则只有当响应内容相对于上次请求改变（根据Last-Modified头的设置）时，请求才被认为是成功的。如果省略，则不会进行头检测。默认为false

（续）

名　称	描　述
jsonp	（字符串）指定一个查询参数名称来覆盖默认的jsonp回调参数名callback
username	（字符串）在HTTP认证请求中使用的用户名
password	（字符串）在HTTP认证请求中使用的密码
scriptCharset	（字符串）当远程和本地内容使用不同的字符集时，用来设置script和jsonp请求所使用的字符集
xhr	（函数）用来提供XHR实例自定义实现的回调函数
traditional	（布尔）如果为true，则使用传统风格的参数序列化。参见第6章中有关$.param()的描述以获取参数序列化的详细信息

要跟踪的选项有很多，但是任何一个请求都不可能用到太多选项。虽然如此，但是当我们计划发起大量请求时，如果能为页面上的这些选项设置默认值岂不是很方便？

8.4.2　设置请求默认值

很明显上一节最后的问题是一个陷阱。你可能已经猜到，jQuery 为我们提供了一种方式来定义一组默认的 Ajax 属性值，如果没有覆盖这些属性的值，则会使用这些默认值。如果要发起很多相似的 Ajax 请求，这可以简化页面。

用来设置 Ajax 默认值列表的函数是$.ajaxSetup()，其语法如下所示。

函数语法：$.ajaxSetup

$.ajaxSetup(options)

把传入的一组选项属性创建为随后调用$.ajax()的默认值

参数

options　（对象）对象实例，其属性定义了一组默认的 Ajax 选项。这些属性与表 8-2 描述$.ajax()函数的属性完全一样。这个函数不应该用来设置成功、出错和完成时的回调处理器（我们将会在 8.4.3 节看到如何使用另外一种替代方法来设置这些回调处理器。）

返回值

未定义

在脚本处理的任意时刻，通常在页面加载时（不过页面开发者可以自由选择任何时间点），可以使用这个函数为随后所有$.ajax()调用设置默认值。

注意　通过这个函数设置的默认值不会应用到 load()方法。对于诸如$.get()和$.post()的实用函数，这些默认值也不会覆盖 HTTP 方法。例如，设置默认 type 为 GET 不会导致$.post()使用 HTTP 的 GET 方法。

8

假设要创建一个包含大多数 Ajax 请求的页面（使用实用函数来创建，而不是 load()方法），我们希望设置一些默认值，以免在每次调用时都指定它们。我们可以在页面头部的<script>元素的第一行编写这样的代码：

```
$.ajaxSetup({
  type: 'POST',
  timeout: 5000,
  dataType: 'html'
});
```

这将会确保随后的每个 Ajax 调用（除了前面提到的 load()方法）都会使用这些默认值，除非使用传入 Ajax 实用函数的属性来显式地覆盖它们。

现在，该怎么处理之前提过的那些通过 global 选项控制的*全局事件*呢？

8.4.3　处理Ajax事件

在执行 jQuery 的 Ajax 请求的过程中，jQuery 会触发一系列的自定义事件，我们可以为这些事件建立处理器，以便跟踪请求的进展，或者在此过程中的不同时间点采取行动。jQuery 将这些事件分类为局部事件和全局事件。

*局部事件*由回调函数来处理，可以直接通过$.ajax()函数的 beforeSend、success、error 以及 complete 选项来指定局部事件，或者间接地通过向便捷方法提供一些回调函数来指定局部事件（反过来，还是使用$.ajax()函数来发起真正的请求）。每当为任何 jQuery 的 Ajax 函数注册回调函数时，我们一直都在处理局部事件，甚至没有意识到局部事件的存在。

*全局事件*是像 jQuery 中其他自定义事件那样被触发的事件，可以通过 bind()方法来为其创建事件处理器（和任何其他事件一样）。这些全局事件（很多都和局部事件相对应）有：ajaxStart、ajaxSend、ajaxSuccess、ajaxError、ajaxStop 以及 ajaxComplete。

全局事件在被触发时会广播到 DOM 中的每个元素上，因此可以在所选择的任何单个或者多个 DOM 元素上创建这些处理器。当执行的时候，处理器的函数上下文被设置为在其上创建处理器的 DOM 元素。

因为无需考虑冒泡层次，所以我们可以在任何元素上创建处理器，以便能通过 this 快速访问此元素。如果不关心特定的元素，那么可以只在<body>上创建处理器，因为没有别的更合适的位置了。但是如果需要对元素做一些特定的处理，比如在 Ajax 请求过程中显示或隐藏动画图片，那么就可以在这个元素上创建处理器，以便通过函数上下文来轻松访问这个元素。

除了函数上下文，还可以通过参数传递给处理器传递其他可用信息。通常来说这些参数有：jQuery.Event 实例、XHR 实例以及传递给$.ajax()的选项。

这个参数列表的异常情况如表 8-3 中所述，该表按照事件触发的顺序列出了 jQuery 的 Ajax 事件。

表8-3　jQuery Ajax事件类型

事件名称	类　型	描　述
ajaxStart	全局	当Ajax请求开始时触发，只要没有其他请求处于激活状态。对于并发的请求，这个事件只会为第一个请求触发 不传递任何参数

（续）

事件名称	类　型	描　述
beforeSend	局部	在发起请求之前调用，以便允许在向服务器发送请求之前修改XHR实例，或者通过返回false来取消请求
ajaxSend	全局	在发起请求之前触发，为了允许在向服务器发送请求之前修改XHR实例
success	局部	当请求返回一个成功的响应时调用
ajaxSuccess	全局	当请求返回一个成功的响应时触发
error	局部	当请求返回一个错误的响应时调用
ajaxError	全局	当请求返回一个错误的响应时触发。传递的第四个参数指向抛出的错误（如果有的话）
complete	局部	当请求结束时调用，而不管状态如何。即便是同步的请求也会调用这个回调函数
ajaxComplete	全局	当请求结束时触发，而不管状态如何。即便是同步的请求也会调用这个回调函数
ajaxStop	全局	当Ajax请求结束并且没有其他并发的请求处于激活状态时触发。不传递任何参数

再次强调（为了清晰起见），局部事件表现为传递给`$.ajax()`（及其相关的便捷函数）的回调函数，而全局事件是被触发的自定义事件，可以通过建立处理器来处理这类事件，就像其他的事件类型一样。

除了使用 `bind()`来创建事件处理器，jQuery 还提供了一组方便的函数来创建处理器，如下所示。

方法语法：jQuery Ajax 事件创建器

`ajaxComplete(callback)`
`ajaxError(callback)`
`ajaxSend(callback)`
`ajaxStart(callback)`
`ajaxStop(callback)`
`ajaxSuccess(callback)`

把传入的回调创建为通过方法名称指定的 jQuery Ajax 事件的事件处理器

参数

callback	（函数）创建为 Ajax 事件处理器的函数。函数上下文（this）是在其上创建处理器的 DOM 元素。可以传入回调的参数如表 8-3 所列

返回值

包装集

下面来完成一个简单的例子，看看如何使用这些事件方法来轻松地跟踪 Ajax 请求的进度。测试页面（太简单了简直不能称之为实验室）的布局如图 8-7 所示，可以通过 URL http://localhost[:8080]/jqia2/chapter8/ajax.events.html 来访问。

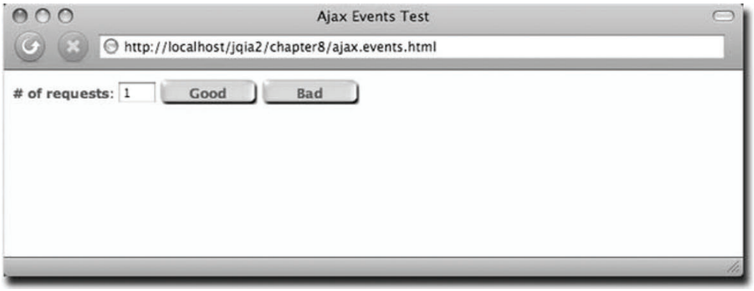


图 8-7　页面的初始布局，用来考察 jQuery Ajax 事件（通过触发多个事件并观察处理器来进行考察）

这个页面展示了三个控件：一个计数字段、一个 Good 按钮以及一个 Bad 按钮。这些按钮用来发出若干请求，请求的个数由计数字段指定。Good 按钮将向有效的资源发出请求，而 Bad 按钮将向无效的资源发出指定个数的以失败告终的请求。

我们也在页面的就绪处理器中创建了很多事件处理器，如下所示：

```
$('body').bind(
  'ajaxStart ajaxStop ajaxSend ajaxSuccess ajaxError ajaxComplete',
  function(event){ say(event.type); }
);
```

这段代码为每一种 jQuery Ajax 事件类型创建处理器，以便向页面“控制台”（放在控件的下方）输出消息，显示被触发的事件类型。

保持请求数为 1，单击 Good 按钮并观察结果，你将会看到每个 jQuery Ajax 事件类型按照其在表 8-3 中描述的顺序依次触发。但是为了理解 `ajaxStart` 和 `ajaxStop` 与众不同的行为，请设置计数控件的值为 2，然后单击 Good 按钮。你会看到页面显示如图 8-8 所示。

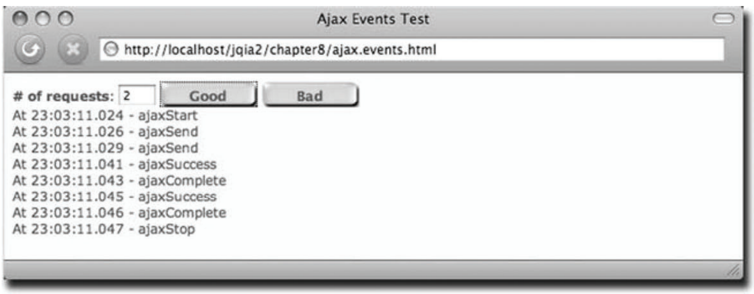


图 8-8　当多个请求处于激活状态时，就会在一组请求而不是单个请求上调用 `ajaxStart` 和 `ajaxStop` 事件

在这里可以看到，当多个请求处于激活状态时，`ajaxStart` 和 `ajaxStop` 事件在一组完整的并发请求上只被触发一次，而其他事件类型会在每个请求上都被触发。

现在尝试单击 Bad 按钮来生成一个无效的请求，然后观察事件行为。

在进入下一章之前，最好先把这些重要的知识应用起来。

8.5 整合所有知识

现在来介绍另一个综合示例。把目前学到的所有知识每样都应用一些：选择器、DOM 操作、高级 JavaScript、事件、特效以及 Ajax。更棒的是，我们将实现另一个自定义 jQuery 方法！

对于这个示例，我们将再次回到 The Boot Closet 页面。为了回顾，请查看图 8-6 来回忆在哪里中断了，因为我们将会继续改善这个页面。

在待售靴子的详细信息中（参见图 8-6），有一些客户可能不熟悉的术语——比如 Goodyear welt 和 stitch-down construction。我们希望使客户能够很容易地了解这些术语的含义，因为知情的客户通常是快乐的客户而快乐的客户乐于掏钱购物！我们喜欢如此。

可以完全照搬 1998 年的设计风格，我们可以提供术语表页面，把用户导航到该页面以便参考，但是这样会使用户的焦点离开我们所期望的地方——客户可以购买东西的页面。我们可以*更加*时尚一点，弹出一个术语提示，甚至显示用户有疑问的术语定义页面。但是即便如此还是有点过时。

如果你再进一步思考，可能就会想知道当客户的鼠标指针在术语上停留时，是否可以使用 DOM 元素的 `title` 特性来显示一个包含术语定义的*提示框*（tooltip，有时也被称为 flyout）。好主意！这就可以就地显示术语定义而无需让客户转移焦点。

但是使用 `title` 特性会带来一些问题。首先，只有当鼠标指针在元素上停留几秒之后才能显示提示框，而我们希望能够更加明显一点，单击术语后立即显示信息。但更重要的是，一些浏览器将会截断 `title` 提示文本，那个限定长度对我们来说太短了。

因此我们要创建自己的提示框！

我们将以某种方式来找到拥有定义的术语，改变它们的显示以便用户能轻松识别这类可单击的元素（赋予这些元素被称为“邀请参与”的样式），设置这些元素，以便鼠标单击元素时会显示一个包含术语描述信息的提示框。随后单击提示框将会从显示器上删除它。

我们还打算把它编写为通用的可重用插件，因此需要确保两件非常重要的事情：

- 没有任何特定于 The Boot Closet 的硬编码；
- 在样式和布局方面赋予页面开发者最大程度的灵活性（在合理的范围内）。

我们将这个新插件称为 Termifier（术语提示器），图 8-9a ~ 图 8-9c 显示的是页面的一部分，显示了我们要添加的行为。



图 8-9a 术语 Full-grain 和 oil-tanned 已经被设置为使用便捷的新插件来添加术语提示行为

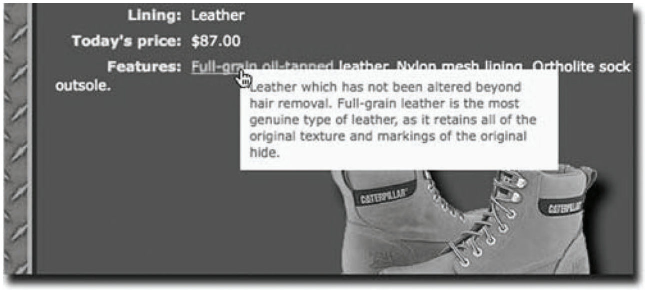


图 8-9b　使用由插件外部的 CSS 指定的简单样式来装扮 Termifier 面板



图 8-9c　拥有更加时尚样式的 Termifier 面板——我们需要给予插件用户这种灵活性

在图8-9a 中，我们看到术语 Full-grain 和 oil-tanned 的描述项目高亮显示。单击 Full-grain 会使包含术语定义的 Termifier 提示框显示出来，如图8-9b 和图8-9c 所示。如图8-9b 中所示，我们提供了一些相当简单的 CSS 样式，而图8-9c 则略显华丽。我们需要确保插件代码允许这种灵活性。

下面就来开始吧。

8.5.1　实现Termifier

你可能还记得，添加 jQuery 方法是通过使用`$.fn`属性来完成的。因为我们已经把新插件称为 Termifier，所以给这个方法起名为`termifier()`。

`termifier()`方法将负责设置匹配集中的每个元素以实现如下目的：

- 在每个匹配元素上建立`click`处理器用来显示 Termifier 提示框；
- 一旦单击元素，将会使用服务器端资源来查找当前元素定义的术语；
- 一旦接收到响应，将会使用淡入特效来在提示框中显示术语的定义；
- 将提示框设置为一旦在其范围内单击就会淡出；
- 服务器端资源的 URL 将会是唯一必需的参数，所有其他的选项都将拥有合理的默认值。

这个插件的语法如下所示。

方法语法：termifier

termifier (url, options)

把包装元素设置为 Termifier 项。为所有的包装元素添加类名 termified

参数

url	（字符串）用来检索术语定义的服务器端动作的 URL
options	（对象）指定下列选项： ❑ paramName——用来把术语发送到服务器端动作的请求参数名称。如果省略，则使用默认值 term ❑ addClass——添加到生成的 Termifier 面板的外部容器的类名。这是除了类名 termifier 之外还要添加的类名，而 termifier 总是会被添加的 ❑ origin——一个包含属性 top 和 left 的散列对象，用来指定 Termifier 面板相对光标位置的偏移量。如果省略，则面板的原点将会被正好放置在光标位置 ❑ zIndex——要赋值给 Termifier 面板的 z-index。默认值为 100

返回值

包装集

我们在名为 jquery.jqia.termifier.js 的文件中创建新 termifier()方法的框架，以便开始实现上述目标：

```
(function($){

  $.fn.termifier = function(actionURL,options) {
    //
    // 实现代码
    //
    return this;
  };

})(jQuery);
```

这个框架使用了第 7 章提到的模式来确保我们可以在实现中自由地使用$，并且可以通过向 fn 原型添加新函数来创建包装器方法。另外要注意如何正确地创建返回值以确保新方法和 jQuery 链能够运行良好。

现在该处理选项了。我们希望将用户提供的选项与自己的默认值合并起来：

```
var settings = $.extend({
  origin: {top:0,left:0},
  paramName: 'term',
  addClass: null,
  actionURL: actionURL
},options||{});
```

之前曾经见过这个模式，因此无需再次讨论它的操作，但是请注意添加 actionURL 参数值到 settings 变量的方式。这将会为即将创建的闭包收集所有稍后要用的值到一个整洁的地方。

收集到所有的数据后，我们现在将继续在包装元素上定义用来创建和显示 Termifier 面板的

8

单击处理器。开始创建该处理器，如下所示：

```
this.click(function(event){
  $('div.termifier').remove();
  //
  // 在此创建新的 termifier
  //
});
```

当单击一个 termified 元素后，我们希望在创建新的 Termifier 面板之前删除任何先前已经存在的面板。否则，这些面板将会散落到屏幕的各个地方，因此要找出所有以前的实例并从 DOM 中删除它们。

注意　你能够想出另外一种永远只显示一个 Termifier 面板的方法吗？

有了这些，我们就可以开始创建 Termifier 面板的结构了。你可能认为我们需要做的所有事情就是创建用来放置术语的定义的单个<div>，尽管这样能够行得通，但也限制了提供给插件用户的选项。考虑图 8-9c 所示的例子，在该例中文本需要与背景“气泡”图片完美地结合在一起。

因此我们将创建一个外部<div>和一个用来存放文本的内部<div>。这不仅仅对于布局有帮助，考虑图 8-10 的情况，在该例中超出定高的结构与文本将会自适应。内部<div>允许页面开发者使用 CSS 规则 overflow 来为提示框文本添加滚动条。



图 8-10　有两个可供操作的<div>元素为页面开发者提供了一些发挥的余地（例如滚动内部文本）

下面来创建外部<div>的代码：

```
$('<div>')                                                   ❶ 创建外部<div>
  .addClass('termifier' +
    (settings.addClass ? (' ') + settings.addClass : ''))    ❷ 添加类名
```

```
.css({                                                  ❸ 设置 CSS 位置
  position: 'absolute',
  top: event.pageY - settings.origin.top,
  left: event.pageX - settings.origin.left,
  display: 'none'
})                                                      ❹ 单击时从显示器上删
.click(function(event){                                    除此元素
  $(this).fadeOut('slow');
})
.appendTo('body')                      ❺ 附加到 DOM 中
```

在这段代码中，我们创建了一个新的<div>元素❶并且调整这个元素。首先，把类名 termifier 赋值给这个元素❷以便以后方便找到它，同时也给予了页面开发者放置 CSS 规则的固定位置。如果调用者提供了 addClass 选项，则也会为元素添加这个类。

然后应用了一些 CSS 样式❸。在这里做的所有工作是使整个功能运转所必需的最低条件（我们允许页面开发者通过 CSS 规则提供额外的样式）。这个<div>元素最初是隐藏的，它被绝对定位在鼠标事件的位置，由调用者提供的任何 origin 来调整。后者允许页面开发者调整位置从而使图 8-9c 中的气泡箭头出现在单击位置。

在那之后，我们建立了一个单击事件处理器❹，用来在单击发生时从显示器上删除元素。最后，元素被附加到 DOM 中❺。

好的，到目前为止一切还不错。现在需要创建内部的<div>（用来包含文本的元素）并把它追加到刚刚创建的元素中，继续编写代码，如下所示：

```
.append(                               ❶ 把内部<div>追加到外部<div>内
  $('<div>').load(
    settings.actionURL,                                 ❷ 获取并向<div>中插入定义
    encodeURIComponent(settings.paramName) + '=' +
      encodeURIComponent($(event.target).text()),       ❸ 提供术语
    function(){
      $(this).closest('.termifier').fadeIn('slow');     ❹ 使 termifier 淡入视野
    }
  )
```

注意，这是创建外部<div>语句的接续——我们不是一直在告诉你 jQuery 链很强大嘛。

在这个代码片段中，我们创建并追加内部<div>❶，发起一个 Ajax 请求来获取并插入术语的定义❷。因为我们使用的是 load()并且想要强制发起 GET 请求，所以需要把参数信息作为文本字符串来提供。在这里不能依赖 serialize()，因为我们没有处理任何表单控件，因此使用 JavaScript 的 encodeURIComponent()方法来自己格式化查询字符串❸。

在请求的完成回调中，我们找到父元素（通过类 termifier 来标记）并把它淡入视野❹。

在庆祝之前，还需要执行最后一个动作才能宣告插件完成。我们必须给包装元素添加类名 termified，以便为页面开发者提供一种改变 termified 元素样式的方式。

```
this.addClass('termified');
```

你瞧！现在我们已经大功告成并且可以享受可口的饮料了。

全部的插件代码如代码清单 8-7 所示，也可以从文件 chapter8/jquery.jqia.termifier.js 中找到。

代码清单 8-7　Termifier 插件的完整实现

```
(function($){

  $.fn.termifier = function(actionURL,options) {
    var settings = $.extend({
      origin: {top:0,left:0},
      paramName: 'term',
      addClass: null,
      actionURL: actionURL
    },options||{});
    this.click(function(event){
      $('div.termifier').remove();
      $('<div>')
        .addClass('termifier' +
          (settings.addClass ? (' ') + settings.addClass : ''))
        .css({
          position: 'absolute',
          top: event.pageY - settings.origin.top,
          left: event.pageX - settings.origin.left,
          display: 'none'
        })
        .click(function(event){
          $(this).fadeOut('slow');
        })
        .appendTo('body')
        .append(
          $('<div>').load(
            settings.actionURL,
            encodeURIComponent(settings.paramName) + '=' +
              encodeURIComponent($(event.target).text()),
            function(){
              $(this).closest('.termifier').fadeIn('slow');
            }
          )
        );
    });
    this.addClass('termified');
    return this;
  };

})(jQuery);
```

这是较为困难的部分。容易的部分应该是在 The Boot Closet 页面应用 Termifier——至少要能够正确使用 Termifier 插件。下面来看看如何使用 Termifier 插件。

8.5.2　测试Termifier插件

因为我们将所有创建和操作 Termifier 提示框的复杂逻辑封装进了 `termifier()`方法，所以在 The Boot Closet 页面使用这个新的 jQuery 方法相对简单。但是首先需要做一些有趣的决策。

例如，我们需要决定如何找出页面上希望添加提示框的术语。记住，我们需要构造一个元素包装集，其中包含了方法将要操作的术语元素。可以使用拥有特定类名的`<span>`元素，代码可能如下所示：

```
<span class="term">Goodyear welt</span>
```

在这种情况下，创建这些元素的包装集会非常简单：`$('span.term')`。

但是有人可能会觉得<span>标签有点冗长。我们将使用很少用到的 HTML 标签<abbr>来取而代它。<abbr>是 HTML 4 添加的标签，用来找出文档中的缩写。因为这个标签纯粹是为了找出文档元素，没有哪个浏览器会在语义或者视觉呈现方面对这些标签做过多的处理，所以特别适合我们使用。

注意 HTML 4[①]还定义了一些语义标签，例如<cite>、<dfn>以及<acronym>。HTML 5 规范草案[②]建议添加更多这样的语义标签，目的是为了提供结构而不是提供布局或者视觉呈现指令。这些标签是<section>、<article>以及<aside>。

因此，我们要做的第一件事情是修改服务器端资源，使其返回包含术语的商品细节，以便将拥有术语表定义的那些术语放置在<abbr>标签中。事实证明，`fetchProductDetails` 动作已经那样做了。由于浏览器不会对<abbr>标签做任何处理，因此除非查看动作文件或者检查动作的响应，否则你可能不会注意到这个标签。一个典型的响应（款式 7141922）包含以下内容：

```
<div>
  <label>Features:</label> <abbr>Full-grain</abbr> leather uppers. Leather
  lining. <abbr>Vibram</abbr> sole. <abbr>Goodyear welt</abbr>.
</div>
```

注意，如何使用<abbr>标签将术语 Full-grain、Vibram 以及 Goodyear welt 识别出来的。

现在回到页面本身。把第三阶段的代码（代码清单 8-6）作为起始点，来看下为了使用 Termifier 需要向页面添加什么。我们需要把新方法引入页面，因此把下面语句添加到<head>节中（在加载 jQuery 之后）：

```
<script type="text/javascript" src="jquery.jqia.termifier.js"></script>
```

当加载商品信息时，需要对添加到页面的任何<abbr>标签应用 `termifier()`方法，因此我们为获取产品详细信息的 `load()`方法添加了一个回调函数。这个回调函数使用 Termifier 来设置所有的<abbr>元素。改进后的 `load()`方法（改变部分用粗体显示）如下所示：

```
$('#productDetailPane').load(
  '/jqia2/action/fetchProductDetails',
  $('#bootChooserControl').serialize(),
  function(){ $('abbr').termifier('/jqia2/action/fetchTerm'); }
);
```

添加的回调函数创建了由所有<abbr>元素组成的包装集并向它们应用 `termifier()`方法，指定了一个服务器端动作 `fetchTerm` 并且让所有其他的选项保持默认值。

这就是全部内容了！（好吧，差不多是了。）

---

① HTML 4.01 规范：http://www.w3.org/TR/html4/。——编者注

② HTML 5 规范草案：http://www.w3.org/html/wg/html5/。——编者注

8

1. 清除无用的东西

因为我们很明智地把所有繁重的工作封装在可重用的 jQuery 插件中，所以在页面上使用这个插件轻而易举！我们也可以轻松地在其他任何页面或者站点上使用这个插件。这才是工程的精髓所在！

但是我们遗漏了一件小事情。我们把显示另一个 Termifier 提示框时删除所有其他提示框的操作封装在插件内部，如果用户选择了一个新的靴子款式，会发生什么呢？哎呀！页面上将会留有一个无关的 Termifier 面板。因此当重新加载产品详细信息时，需要删除任何已经显示的 Termifier 面板。

可以只添加一些代码到 `load()`回调函数，但这种方式似乎是错误的，因为它会将 Termifier 和产品详细信息的加载耦合在一起。如果能够保持这两者解耦合，而且只需监听那些通知什么时候可以删除任何 Termifier 的事件，那么我们会更加高兴。

如果想到 `ajaxComplete` 事件，可以吃一盒枫糖核桃冰激凌或者其他喜欢的食物奖励一下自己。可以监听 `ajaxComplete` 事件，当此事件是和 `fetchProductDetails` 动作绑定时删除任何现有的 Termifier：

```
$('body').ajaxComplete(function(event,xhr,options){
 if (options.url.indexOf('fetchProductDetails') != -1) {
   $('div.termifier').remove();
 }
});
```

现在来看下如何应用各种样式到 Termifier 提示框。

2. 为提示框应用样式

为元素应用样式是件相当简单的事情。在样式表中我们可以轻松地应用规则从而使（应用 `termifier` 插件后的）术语以及 Termifier 面板就像图 8-9b 所示的那样。查看 `bootcloset.css`，可以看到以下代码：

```
abbr.termified {
  text-decoration: underline;
  color: aqua;
  cursor: pointer;
}

div.termifier {
  background-color: cornsilk;
  width: 256px;
  color: brown;
  padding: 8px;
  font-size: 0.8em;
}
```

这些规则给予了术语类似链接的外观从而吸引用户来单击术语，并且还给予了 Termifier 提示框如图 8-9b 所示的简单外观。页面的这个版本可以在 http://localhost[:8080]/jqia2/chapter8/bootcloset/phase.4a.html 上找到。

将 Termifier 面板带入下一个层级，如图 8-9c 所示，就要灵活使用插件中提供的一些选项。对于更加时尚的版本，可以通过下面的代码来调用 Termifier 插件（在 `load()`回调函数中）：

```
$('#productDetailPane').load(
  '/jqia2/action/fetchProductDetails',
  $('#bootChooserControl').serialize(),
  function(){ $('abbr')
    .termifier(
      '/jqia2/action/fetchTerm',
      {
        addClass: 'fancy',
        origin: {top: 28, left: 2}
      }
    );
  }
);
```

这个调用和前一个示例的不同之处仅在指定了要添加到 Termifier 的类名 `fancy`，调整了提示框的原点以便使气泡的尖端出现在鼠标事件的位置。

向样式表添加如下规则（同时保持原有的规则不变）：

```
div.termifier.fancy {
  background: url('images/termifier.bubble.png') no-repeat transparent;
  width: 256px;
  height: 104px;
}

div.termifier.fancy div {
  height: 86px;
  width: 208px;
  overflow: auto;
  color: black;
  margin-left: 24px;
}
```

这会添加如图 8-9c 所示的所有时尚的样式。

这个新页面可以在 http://localhost[:8080]/jqia2/chapter8/bootcloset/phase.4b.html 上找到。新插件不仅有用而且强大，不过总是有提高的余地。

8.5.3 改进Termifier

新出炉的 jQuery 插件虽然相当好用，但是还有一些小问题和可以进行大幅度改进的潜力。为了磨练你的技能，下面的列表列出了可以对 `termifier()`方法或者 The Boot Closet 页面做的改进。

- 添加一个选项（或者若干选项）来允许页面开发者控制消退效果的持续时间，或者允许他使用另一种特效。
- 在客户单击 Termifier 提示框、显示另一个提示框或者重新加载产品详细信息页面之前，Termifier 提示框会一直显示。为 Termifier 插件添加一个超时选项，以便在超时时间过期后自动使提示框消失（如果提示框仍然处于显示状态）。
- 单击提示框来关闭它，这个操作会带来易用性问题，因为不能选择提示框中的文本来剪切粘贴。修改插件以便用户单击页面上提示框之外的任何地方都会关闭提示框。

- 我们没有在插件中做任何错误处理。如何改进代码来优雅地处理无效的调用信息或者服务器端的错误呢？
- 我们使用部分透明的 PNG 文件来在图片中实现吸引人的阴影。尽管大部分浏览器能很好地处理这种文件格式，但 IE 6 却不能处理 PNG 文件并且会为其显示白色的背景。为了解决这个问题，我们也可以为图片提供没有阴影的 GIF 格式。尽管对 IE 6 的支持逐渐减少（事实上，Google 已经在 2010 年 3 月 1 日放弃了对 IE 6 的支持），如何改进页面以便检测出何时使用 IE 6 从而替换所有的 PNG 引用为相应的 GIF 图片？
- 说到图片，每种靴子款式只有一张照片而已，即使该款式有多种颜色可供选择。假设每个可能的颜色都有相应的照片，如何改进页面以便在颜色改变时显示对应的图片呢？

你还能想出可以对这个页面或者 `termifier()`插件做的其他改进吗？请在本书的论坛分享你的想法以及解决方案，论坛的网址是：http://www.manning.com/bibeault2。

8.6　小结

毫不奇怪，这是本书中最长的一章。Ajax 是新一波 DOM 脚本应用程序的关键部分，jQuery 也非常乐意提供一组丰富的工具集供我们使用。

对于加载 HTML 内容到 DOM 元素，`load()`方法提供了一种简单的方式来从服务器获取内容并使之成为任意包装集元素的内容。使用 `GET` 还是使用 `POST` 方法取决于如何提供传递到服务器的参数数据。

当需要 `GET` 方法时，jQuery 提供了实用函数`$.get()`和`$.getJSON()`；如果从服务器返回 JSON 数据，那后者会很有用。为了强制发起 `POST` 方法，可以使用`$.post()`实用函数。

如果需要最大的灵活性，那么可以使用带有丰富分类选项的`$.ajax()`实用函数，它允许我们控制 Ajax 请求的大部分细节。jQuery 中所有其他的 Ajax 特征都使用这个函数的服务来提供自己的功能。

为了使得管理大量的选项不那么繁琐，jQuery 提供了`$.ajaxSetup()`实用函数，允许为`$.ajax()`函数（包括使用`$.ajax()`服务的所有其他的 Ajax 函数）的任何常用选项设置默认值。

为了完善 Ajax 工具集，jQuery 也允许我们通过在各个阶段触发 Ajax 事件来监听 Ajax 请求的过程，并允许建立处理器来监听这些事件。我们可以对这些处理器使用 `bind()`方法，也可以使用以下便捷方法：`ajaxStart()`、`ajaxSend()`、`ajaxSuccess()`、`ajaxError()`、`ajaxComplete()`以及 `ajaxStop()`。

在掌握了这些令人印象深刻的 Ajax 工具集后，在 Web 应用程序中启用丰富的功能会很容易。记住，如果有的东西 jQuery 没有提供，那么利用 jQuey 的现有特征来扩展 jQuery 也是非常容易的。或者可能已经存在这样一个插件（官方的或者非官方的）正好可以满足你的需求！

Part 2 第二部分

jQuery UI

本书的第一部分主要介绍 jQuery 核心库，我们做了许多工作来学习如何轻松地扩展 jQuery。这么做是正确的，因为扩展堪称 jQuery 的最大特点。它拥有的各种官方插件、大量可用的非官方插件以及伴随的 jQuery UI 库无疑都证明了这点。

jQuery UI 是建立在核心库功能基础上的，为创建美观大方、一目了然的用户界面提供了大量高级组件。

我们将从学习如何获取和配置库开始——这不像复制单个文件（就像对核心库进行的操作那样）那样简单。然后我们还会看一些 jQuery UI 向核心库特征中添加的基础功能。

由此我们将看到 jQuery UI 是如何在核心库之上搭建一层新框架的，还会看到这层框架添加的扩展功能，这些功能给我们带来了用户界面交互，从而可以拖放元素、排序元素以及缩放元素。不仅如此，jQuery UI 又以这层框架为基础，为我们提供了一些用户界面部件（widget）。我们也将详细研究这些部件。

完成了这个部分，也就是完成了对本书的学习之后，你就能去完成任何 Web 用户界面项目。下面就开始学习吧！

第 9 章

jQuery UI 简介：主题和特效

本章内容

- ❑ jQuery UI 概述
- ❑ 配置和下载 jQuery UI 库
- ❑ 获取和创建 jQuery UI 主题
- ❑ 扩展 jQuery UI 提供的特效
- ❑ 其他对核心库的扩展

jQuery UI 不只是一个插件，但却不属于 jQuery 核心库，它是 jQuery 核心库的一个官方扩展，旨在为启用 jQuery 的 Web 应用页面提供扩展用户界面（UI）的能力。

基于浏览器环境中使用的工具（如 JavaScript、DOM 操作、HTML，甚至是核心 jQuery 库）给予了我们一些基本能力，几乎可以将所有想要提供给用户的任意类型的交互组件组装在一起。但即便如此，使用基本构造块来创建复杂的交互仍然令人生畏。用来进行 DOM 操作的原生 JavaScript API 极度乏味（幸运的是我们有核心 jQuery 库来对付它），而且与桌面应用程序相比，HTML 提供的几个表单控件也实在太少了。

我们可以使用目前为止学到的 jQuery 方法来创建自己的交互组件和控件（也称为部件）。但是 jQuery UI 库提供了许多万众期待的扩展特征。jQuery UI 提供了一些高级组件，来帮助创建这些交互组件和控件。

设想一个常用的部件例如进度条。我们可以分析其需求，然后想想如何使用核心 jQuery 来实现它，不过 UI 库已经预料到了这个需求并且提供了一个现成的进度条部件。

与 jQuery 核心库不同，jQuery UI 是松散耦合元素的联合体。我们将会在 9.1 节看到，如何下载一个包含所有元素的库，或者下载仅仅包含需要的元素的库。这些元素大体分为以下 3 类。

- ❑ 特效类——除了 jQuery 核心库提供的特效之外的增强特效。
- ❑ 交互类——鼠标交互，比如拖放元素、排序元素等。
- ❑ 部件类——一组常用的控件，比如进度条、滑动条、对话框、选项卡等。

值得注意的是，交互组件和 UI 部件使用了大量的 CSS 来“布置”可见元素。这是使元素正确工作的基本工具，也使其符合页面的设计和外观，本章稍后将会专门探讨这个话题。

jQuery 的内容很多。因为 jQuery UI 是 jQuery 的一个重要扩展，所以我们用 3 章的篇幅来探

讨它。我们也提供了一组 UI 实验室页面——每个页面几乎都对应着 jQuery UI 的一个主要领域。这几章的内容连同实验室页面，应能帮助你轻松走进 jQuery UI 的世界。

闲话少说，下面就开始着手学习 jQuery UI 吧。

9.1　配置并下载 jQuery UI 库

jQuery UI 库由很多元素组成。根据应用程序的需求，你可能会用到所有元素，也可能只使用它们的一个子集。例如，你的应用程序可能不需要使用部件，但是它可能需要拖放元素的功能。

jQuery UI 团队提供了构建库文件的功能，这个库文件可以仅由基本部件组成，也可以加上你想要为应用程序添加的任何功能。这就避免了加载一个规模庞大，还包含很多多余功能的库。毕竟，为什么要在所有页面上加载一大堆无用的脚本呢？

9.1.1　配置和下载库

在使用 jQuery UI 库之前，我们需要下载它。jQuery UI 的下载页面在 http://jqueryui.com/download，如图 9-1 所示。从图中可以看到，编写此书时 jQuery UI 的最近版本是 1.8。

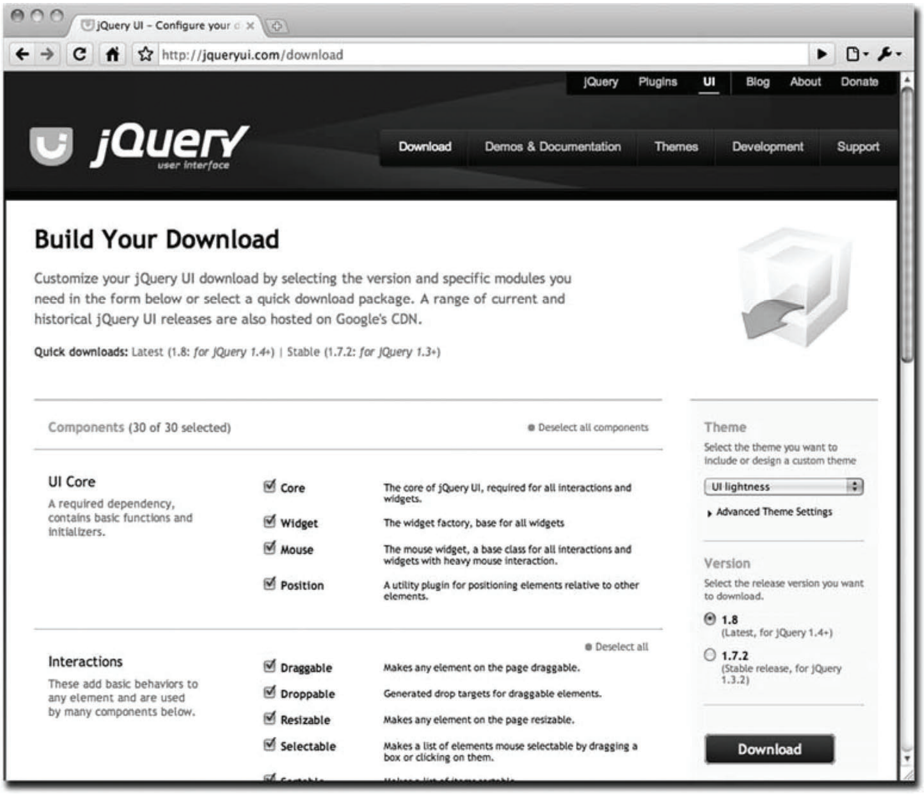


图 9-1　jQuery UI 下载页面允许我们配置和下载一个根据应用程序需求来定义的 jQuery UI 库

在下载页面上，你会看到 jQuery UI 可用的组件列表，勾选组件对应的复选框就可以选择该组件。你最好勾选 UI Core 的复选框，因为所有交互组件和大部分部件都会用到它。但是不要为选择哪些选项过度纠结，这个页面会自动选择从属组件，并且不会让你将无效的组件放在一起。

选好组件后（目前，我们建议你出于学习的目的选择所有组件），可从最右侧一栏的下拉列表中选择一个主题，然后单击 Download 按钮。

目前来说选择哪个主题并不重要，稍后我们将在本章中阐述 CSS 主题。现在，请任意选择主题，但目前不要选择 No Theme（无主题）选项。你肯定想要下载 CSS 主题而不是从头创建它。相信我——你随时可以替换或者调整 CSS 主题。

注意　选择的主题对所选组件生成的 JavaScript 代码没有任何影响。主题之间的区别只限于样式表和主题相关的图片。

本书示例代码所提供的 jQuery UI 库配置包含了所有组件并且使用了 Cupertino 主题。

9.1.2　使用jQuery UI库

单击 Download 按钮，一组压缩的自定义 jQuery UI 库文件就会被下载到你的系统中（下载到哪里取决于浏览器设置）。压缩文件包含如下文件和文件夹。

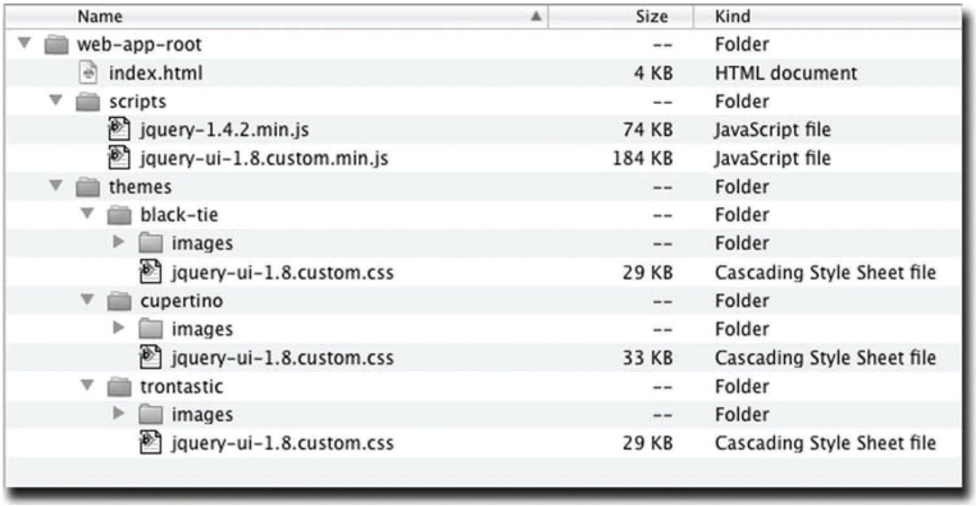
- index.html——HTML 文件，包含所下载的部件示例，这些部件是通过选中的主题渲染的。你可以使用该页面快速检查以确定是否包含了全部想要的部件，以及主题是否和你想象的完全一致。
- css——文件夹，其下的子文件夹包含选中主题的 CSS 文件和图片。子文件夹以选中主题命名，例如 cupertino 或者 trontastic。
- development-bundle——包含开发者资源的文件夹，比如许可文件、示例、文档以及其他有用的文件。在你空闲的时候多熟悉下这个文件夹，其中有很多好东西。
- js——文件夹，包含生成的 jQuery UI 的 JavaScript 代码，以及一个核心 jQuery 库的副本。

要使用这个库，你需要将 CSS 文件夹里的主题文件夹、js 文件夹里的 jQuery UI 库文件 jquery-ui-1.8.custom.min.js 复制到 Web 应用程序里的相应位置。如果尚未包含核心库，那还需要复制 jQuery 核心库。

注意　JavaScript 文件的名称将会反映 jQuery UI 的当前版本，因此它将会随着 jQuery UI 的更新而改变。

尽管可以根据 Web 应用程序的需求来指定放置这些文件的位置，但是保持主题的 CSS 文件与其图片的关联是很重要的。除非你想要改变 CSS 文件里的所有图片的引用，否则务必确保主题图片和 CSS 文件是放置在同一个文件夹中的。

支持多个主题的常用应用程序布局如图 9-2 所示。在这里，我们支持三套打包好的可供下载的主题。



Name	Size	Kind
web-app-root	--	Folder
index.html	4 KB	HTML document
scripts	--	Folder
jquery-1.4.2.min.js	74 KB	JavaScript file
jquery-ui-1.8.custom.min.js	184 KB	JavaScript file
themes	--	Folder
black-tie	--	Folder
images	--	Folder
jquery-ui-1.8.custom.css	29 KB	Cascading Style Sheet file
cupertino	--	Folder
images	--	Folder
jquery-ui-1.8.custom.css	33 KB	Cascading Style Sheet file
trontastic	--	Folder
images	--	Folder
jquery-ui-1.8.custom.css	29 KB	Cascading Style Sheet file

图 9-2　使用 jQuery UI 的应用程序其脚本和主题文件的常用布局（该布局可能因人而异）

切换主题非常轻松，只需要改变应用程序页面中引用 CSS 文件的 URL 即可。

文件就位之后，就可以使用`<link>`和`<script>`标签将它们导入到页面上。例如，可以使用如下标记将文件导入到 index.html 文件，以实现图 9-2 所描述的应用程序布局：

```
<link rel="stylesheet" type="text/css"
      href="themes/black-tie/jquery-ui-1.8.custom.css">
<script type="text/javascript" src="scripts/jquery-1.4.2.min.js"></script>
<script type="text/javascript"
        src="scripts/jquery-ui-1.8.custom.min.js"></script>
```

切换主题可以通过改变`<link>`标签里的主题名称来实现。

注意　如果你想让用户很容易地动态切换主题，可以试试 http://jqueryui.com/docs/Theming/ThemeSwitcher 上的 Theme Switcher Widget。

有了这些知识在手，我们就具备了探索 jQuery UI 的能力。在本章的后续几节中，我们将会从仔细研究主题开始，然后看看 jQuery UI 扩展核心库的方法和功能，尤其是在特效领域。在第 10 章和第 11 章，我们将会首先探索鼠标交互然后再研究部件。

9.2　jQuery 的主题和样式

jQuery UI（特别是部件集）高度依赖于下载的 CSS 文件里定义的 CSS 类，这个 CSS 文件是用来设置页面上可见元素的样式。我们将会在本章以及接下来的两章中学到，jQuery UI 对赋给

元素的类名的依赖，要比对样式的依赖要多得多。

设置 jQuery UI 所用主题有很多方式。这里列出了这些方式，由易到难。

- 在下载过程中选择一个主题，并原封不动地使用它。
- 使用 ThemeRoller Web 应用程序来设计自己的主题。我们将在 9.2.2 节中快速介绍 ThemeRoller 的用法。
- 通过修改原始的 CSS 文件，或者提供一个 CSS 文件覆盖原始设置来调整下载的主题。
- 从头编写自己的主题。

不推荐采用最后那种方式。这需要定义的大量的类（预定义主题的 CSS 文件有 450 ~ 500 行之多），一旦出错，后果会非常严重。

下面就从如何组织预定义 CSS 文件和类名来开始吧。

9.2.1　概述

尽管预定义主题看起来很美好，但是几乎找不到一个能精确匹配 Web 应用程序的主题。

当然也可以先下载一个现成的主题，然后将其作为网站的最终外观，但是这样的好事并不常有。我们中的每个人都可能会遇到这种情况，市场经理或者产品经理站在我们后面问道："把它变成蓝色会怎么样呢？"

下一节将要讨论的 ThemeRoller 可以帮助构造一个主题，使其满足我们需要的颜色和外观，但即便如此，我们可能仍然需要逐页调整。因此，了解 jQuery UI 是如何组织和使用 CSS 类是非常必要的。

下面从考察如何命名这些类开始。

1. 类的命名

jQuery 定义和使用的类名非常广泛，但结构清晰合理。命名类时考虑周到，因为类名不仅传达了类的含义，而且还说明了在哪里和如何使用类的信息。尽管名称很多，但是如果你掌握了类名构建的窍门，就会发现其中的逻辑，从而轻松地管理它们。

首先，为了防止在类的命名空间里使用其他人的命名，所有的 jQuery UI 类名都是以 `ui-`作为前缀。类的名称都是小写的，并使用连字符（-）来分隔单词，例如：`ui-state-active`。

在 jQuery 库中，有些类随处可见。前面提及的类 `ui-state-active` 就是一个很好的例子。所有 UI 库的组件都使用它来指示一个元素处于激活状态。例如，Tab 部件用它来标记当前选中的选项卡，而 Accordion 部件用其来找出打开的可折叠面板。

有些类是某些组件（如交互元素和 UI 部件）所特有的。这时，组件名称会直接出现在 `ui` 前缀之后。例如，属于 Autocomplete 部件的类都会以 `ui-autocomplete` 开头，而属于 `Resizable` 交互部件的类都会以 `ui-resizable` 作为开头。

在本节的剩余部分，我们将会仔细考察可用于多个库的类的分组。在接下来的几章中对各种组件进行考察时，我们将会讨论那些组件所特有的类。我们不可能讨论所有 jQuery UI 定义类而是会探讨最重要的、可能需要在页面上使用的类。

2. 识别部件

用 jQuery UI 库来创建部件时，构成部件的元素可以由 jQuery 库来创建，也可以由已经存在于页面上的现有元素来创建。

jQuery UI 使用一组以 `ui-widget` 开头的类名来标识构成部件的元素。类 `ui-widget` 用来标识部件的主元素——通常是一个容器，它是构成部件的所有元素的父元素。

其他类名，例如 `ui-widget-header` 和 `ui-widget-content` 则酌情由部件的相应元素使用。各个部件使用这些类的方式可能不尽相同。

3. 状态跟踪

在不同的时间点，部件或交互元素的各个部分可能处于不同的状态。jQuery UI 通过一组以 `ui-state` 开头的类来跟踪这些状态，并为其应用合适的视觉样式。这些状态包括 `ui-state-default`、`ui-state-active`、`ui-state-focus`、`ui-state-highlight`、`ui-state-error`、`ui-state-disabled` 以及 `ui-state-hover`。

我们可以在自己的脚本或 CSS 文件里使用上述名称来跟踪状态或改变各种状态下的元素样式。

4. 图标

jQuery UI 定义了大量的图标可供各种部件使用。例如，在 Tab 部件的选项卡元素上用图标指示状态，或者直接在 Button 部件上放置的图标。每个图标都以 `ui-icon` 开头的类名来标识。例如，`ui-icon-person`、`ui-icon-print` 以及 `ui-icon-arrowthick-1-sw`。

jQuery UI 处理图标的方式非常智能。每个图标都被定义在一张单个图片的网格上，称为图标簿（如果你也同意这种称呼的话）。这样一来，只要浏览器下载并缓存这张图片，就不需要再访问服务器来显示任何可用的图标了——这些图标数量众多（写作此书时已有 173 个图标）。定义图标类的目的是将这个原始的图标簿作为背景图片来移动，从而使得所需的图标显示在元素的背景上。

我们将会在第 11 章更详细地探讨图标以及 Button 部件，但是如果你想先睹为快，可以在浏览器中打开文件 chapter11/buttons/ui-button-icons.html。

5. 圆角

如果已经看过 jQuery UI 定义的部件，那么你可能已经见过很多圆角效果。

jQuery UI 使用了一组类，它们定义了适合特定浏览器和 CSS3 的样式规则，以便在所支持的浏览器中显示圆角。不支持圆角的浏览器不会显示圆角。

这些圆角类并非只为 jQuery UI 部件而设。我们可以在页面的任何元素上使用这些类。

用浏览器打开文件 chapter9/lab.rounded-corners.html，打开迷你圆角实验室页面（这个页面太简单了，称不上一个完整的实验室页面）。你将会看到如图 9-3 所示的页面。



图 9-3　迷你圆角实验室向我们展示了如何通过简单的类赋值来将圆角效果应用到元素上

复选框控件允许我们选择要应用 `ui-corner` 类中的哪些类（如果有的话）到测试对象上。勾选一个复选框，就会对其应用相应的类名；取消勾选，就会删除该类名。

注意　在 IE8 或更早的版本不支持任何类型的圆角（是从 IE 9 才开始支持圆角），并且 Firefox（至少在写本书的时候）不支持图片元素上的圆角。

花一些时间去单击不同的复选框来，看看类的应用是如何改变测试对象的各个拐角的。

9.2.2　使用ThemeRoller 工具

如果在下载 jQuery UI 后，你快速浏览过生成的 CSS 文件，可能很快就会得出一个结论：试图从头编写这样的文件简直是疯狂的行为。看一眼 CSS 相应的图片就能证明这一点。

如果主题包与站点的需求不匹配，还有以下这些明智选择：

- 选择一个最接近需求的主题包，然后调整到所喜欢的样子；
- 使用 ThemeRoller 工具，从头创建一个主题。

结果证明，ThemeRoller 工具是完成上述任何一种选择的最好方法。使用 ThemeRoller 工具，我们可以利用它简洁直观的界面从头指定主题的所有细节，也可以预加载一个预定义的主题，再将其调整到我们喜欢的样子。

ThemeRoller 工具如图 9-4 所示，可以从 http://jqueryui.com/themeroller/获取。

我们不会过多介绍使用 ThemeRoller 的细节，因为弄明白这些细节相当容易。但还是有一些事情，值得你花些时间去了解。

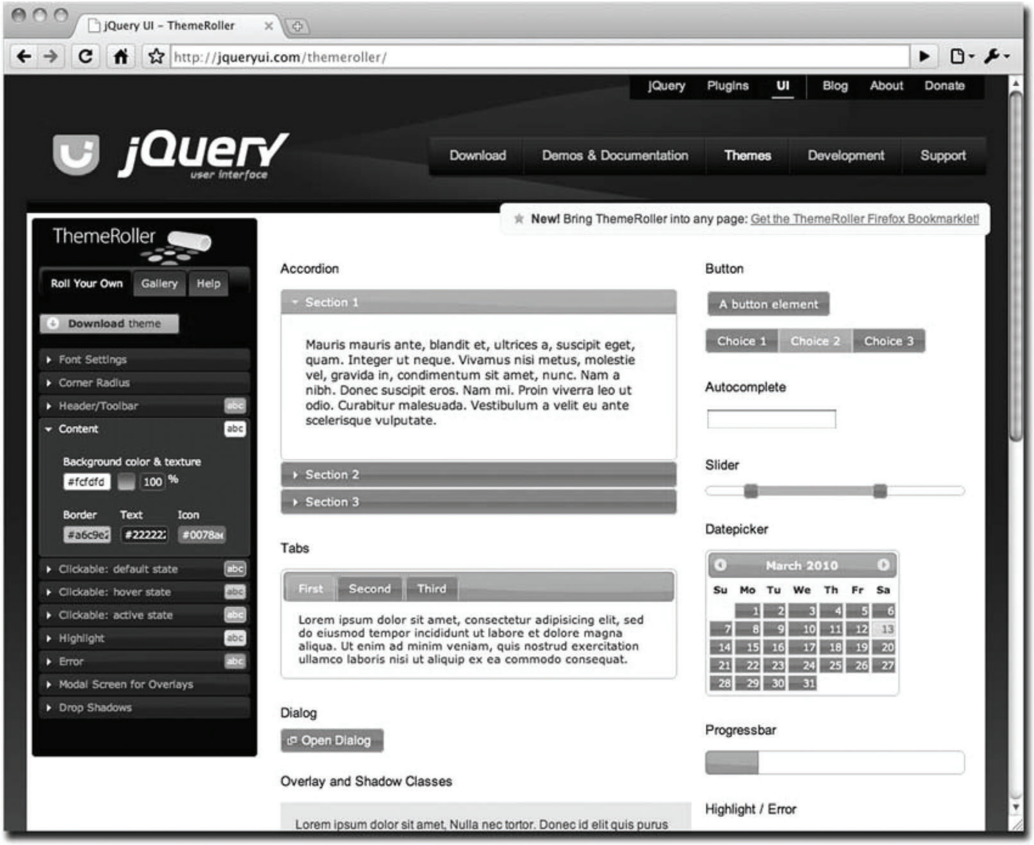


图 9-4 jQuery UI 的 ThemeRoller 工具使用简洁直观的界面来创建交互式自定义主题

1. ThemeRoller 的基本用法

界面左边的 ThemeRoller 控制面板有三个选项卡，如下所示。

- ❑ Roll Your Own——我们将在这里完成大部分的工作。各种各样的面板（单击一个面板标题即可打开该面板）可用来指定主题的所有细节。显示区域里的变化实时更新，以便展示对设置的更改是如何影响各个部件的。
- ❑ Gallery——列出了所有包含的主题包。我们将会在下节中讨论这个列表。
- ❑ Help——当你遇到困难时可参考的帮助文本。

当对自己的主题满意的时候，单击 Roll Your Own 选项卡上的 Download Theme 按钮就可以转到 Build Your Download 页面，这样就可以下载自定义主题了。（主题的设置作为请求的参数，通过 URL 来传递。）

我们在 9.1 节介绍过，单击 Build Your Download 页面上的 Download 按钮可以下载主题。

2. 从主题包开始

通常来说，与完全从头创建一个自定义主题相比，一个现成的主题包可能更容易起步。如果

你想加载一个预定义主题的设置，并对其进行调整，请遵循以下这些简单的步骤。

(1) 选择 Gallery 选项卡。

(2) 仔细了解预定义主题，然后选择你想要开始设置的主题。单击该主题，显示区域里的部件将会展示该主题的设置。

(3) 返回到 Roll Your Own 选项卡。注意，预定义主题的设置已经被设定到控件里了。

(4) 根据你自己的意愿调整主题的设置。

(5) 主题调整完成后单击 Download Theme 按钮。

下载的 CSS 文件和图片将会反映出你为自定义主题所选择的设置，在下载的内容中，CSS 文件夹内包含该主题的文件夹将会被命名为 custom-theme。

3. 重新加载主题

当你正在为 Web 应用程序里应用了自定义主题而自鸣得意时，总会有人走过来对你的设置指手画脚。虽然经过努力尝试，但是你在 ThemeRoller 上却找不到上传控件或者重新加载控件。为了修改主题，真的需要每次都重新创建一个自定义主题吗？当然不需要。

在下载的 CSS 文件里（大约在第 44 行，不是在最顶端），你将会找到一句包含文本“* To view and modify this theme visit”的注释，后面跟着一个相当长的 URL。把这个 URL 复制粘贴到浏览器地址栏中，它将会带你转到 ThemeRoller 页面，该页面上已经加载了自定义主题的设置（自定义主题作为请求的参数被编码到 URL 上）。可以任意修改主题的设置，完成后下载新的文件。

好了，现在一个 jQuery UI 的主题已经安装完毕、整装待发。下面就进入 jQuery UI 提供的扩展特效吧。

9.3　jQuery UI 特效

在第 5 章，我们介绍了使用 jQuery 动画引擎来创建自定义特效是多么简单。jQuery UI 利用核心动画引擎来为我们提供了一套丰富的现成特效，其中一些还实现了自定义特效（我们作为练习而创建的特效）。

我们将会仔细介绍这些特效，也会看到 jQuery UI 是如何通过提供（通常不支持特效的）核心方法的扩展版本来将这些特效注入核心 jQuery 库的。也会看到一些 jQuery UI 提供的专注于特效的新方法。

我们先来看下这些特效。

9.3.1　jQuery UI特效

jQuery UI 提供的所有特效都可以通过 `effect()`方法来独立使用，无需再用其他方法。`effect()`方法可以用来开启包装集元素上的特效。语法如下所示。

方法语法：effect

effect (type, options, speed,callback)

执行包装集元素上的指定特效

参数

type	（字符串）要运行的特效。为以下参数之一：blind、bounce、clip、drop、explode、fade、fold、highlight、puff、pulsate、scale shake、size、slide 以及 transfer。参见表 9-1 来了解这些特效类型的详细信息
options	（对象）提供核心 animate()方法定义的指定特效的选项（参见第 5 章）。此外，每个特效都有自己的一组选项，可以指定跨多个特效的共用选项，参见表 9-1
speed	（字符串|数字）提供 slow、normal、fast 参数之一或者以毫秒为单位的特效持续时间。（可选）如果省略，则默认为 normal
callback	（函数）一个可选的回调函数，在特效结束后为每个元素所调用。不向此函数传递任何参数，设置函数的上下文为执行动画的元素

返回值

包装集

尽管表 9-1 试图描述每个特效的行为，但实际上直接运行特效会了解得更清楚。建立 jQuery UI 特效实验室就是为了这个目的。这个实验室页面如图 9-5 所示，也可以从 chapter9/lab.ui-effects.html 找到该页面。

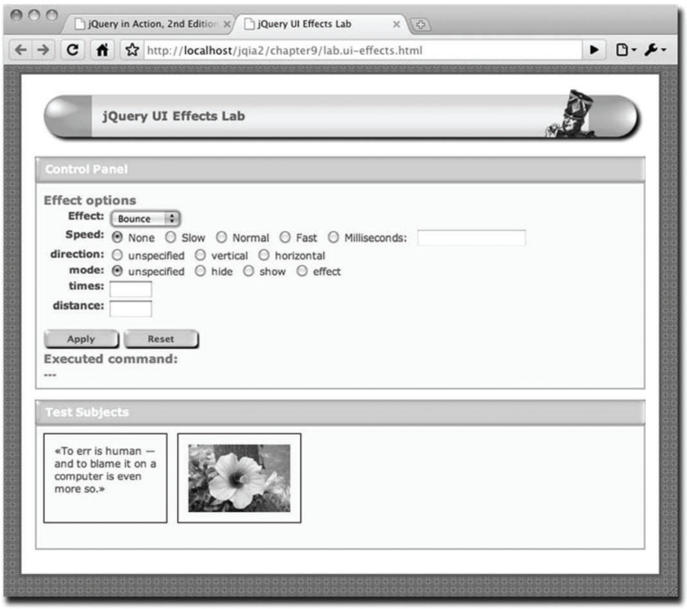


图 9-5 jQuery UI 特效实验室允许我们通过不同的选项来实时地观察 UI 特效的运行效果

这个实验室允许我们在运行中观察每个特效。选中了一个特效后，可以指定的特效选项如控制面板所示（使用的当然是 jQuery UI 特效）。在阅读表 9-1 中的特效说明时，请使用这个实验室来仔细观察每个特效能做什么，以及选项如何影响特效的运行效果。

表 9-1 中描述了各种特效及其选项。所有的特效都接受一个 easing 选项，用来指定要使用特效的缓动函数。在接下来的 9.3.5 节中我们将会探讨缓动的概念。

在阅读表 9-1 中的条目时，请使用 jQuery 特效实验室在运行中观察每个特效。

回想在第 5 章介绍的 animate()方法，这个方法通过逐步改变 CSS 属性值来实现动画。你可能会想到，颜色属性是不支持动画的。

那么，jQuery UI 实现了用来对元素的背景颜色应用动画的 highlight 特效了吗？一起来找找吧。

表9-1　jQuery UI 特效

特效名称的描述	指定特效的选项
blind 以百叶窗的方式显示或者隐藏元素：通过向下或向上移动下边缘，或者向右或向左移动右边缘来实现，具体的实现方式取决于指定的选项direction和mode	direction：（字符串）horizontal或vertical两者之一。如果省略，则默认值为vertical mode：（字符串）show或hide（默认值）两者之一
bounce 使元素在垂直或者水平方向弹跳，可以选择是显示还是隐藏元素	direction：（字符串）up、down、left或right其中之一，如果省略，则默认值为up distance：（数字）以像素为单位的弹跳距离。默认为20个像素 mode：（字符串）effect、show或hide其中之一。如果省略，则使用effect，这会简单地使元素在原地弹跳，而不会改变元素的可见性 times：（数字）弹跳的次数。如果省略，则默认为5
clip 通过将元素的两条相对的边框移动直至在中间相遇，或反之从中间向外移动来显示或者隐藏元素	direction：（字符串）horizontal或vertical两者之一。如果省略，则默认值为vertical mode：（字符串）show或hide（默认值）其中之一
drop 通过使元素看起来像是掉入或者掉出页面来显示或者隐藏元素	direction：（字符串）left（默认值）、right、up或down其中之一 distance：（数字）移动元素的距离。默认为高度或者宽度的一半，根据特效移动元素的方向而定 mode：（字符串）show或hide（默认值）其中之一
explode 通过将元素分解成很多碎片并沿着射线方向移动（就像是元素碎片在页面上聚合到一起或者发散出来）来显示或者隐藏元素	mode：（字符串）show、hide或toggle其中之一 pieces：（数字）在特效里使用的碎片个数。如果省略，则默认值为9。注意，采用的算法可能会为了提高性能而改变这个数字
fade 通过调整元素的不透明度来显示或者隐藏元素。这与核心的渐隐特效一样，但没有选项	mode：（字符串）show、hide（默认值）或toggle其中之一

（续）

特效名称的描述	指定特效的选项
fold 通过分开或者合并相对的一组边框，然后再对另一组边框做相同的操作来显示或者隐藏元素	horizFirst：（布尔）如果是true，就先移动水平的边框。如果省略或指定为false，则先移动垂直边框 mode：（字符串）show或hide（默认值）其中之一 size：（数字）“已折叠”元素的尺寸，以像素为单位。如果省略，则将尺寸设置为15个像素
highlight 在显示或者隐藏元素时通过立即改变元素的背景颜色来吸引注意力	color：（字符串）用作高亮的颜色。可以通过CSS颜色名称例如orange、十六进制符号例如#ffffcc或#ffc，或者三元组的RGB例如rgb(200,200,64)来表示 mode：（字符串）show（默认值）或hide两者之一
pulsate 在确定显示还是隐藏元素之前调高或者调低元素的不透明度	mode：（字符串）show（默认值）或hide两者之一 times：（数字）元素闪烁的次数，默认为5
puff 在调整元素的不透明度时原地放大或缩小元素	mode：（字符串）show或hide（默认值）两者之一 percent：（数字）puff特效的目标百分比。默认为150
scale 通过一个百分比来放大或缩小元素	direction：（字符串）horizontal、vertical或both其中之一。如果省略，则默认为both fade：（布尔）如果指定为true，则根据是显示还是隐藏元素来同时调整不透明度和尺寸 from：（对象）一个对象，其height和width属性指定初始尺寸。如果省略，则从元素当前尺寸开始变化 mode：（字符串）show、hide、toggle或effect（默认值）其中之一 origin：（数组）如果mode不是effect，则为特效定义基础消失点，指定为包含两个字符串的数组。可设置的值有：top、middle、bottom和left、center、right。默认为['middle','center'] percent：（数字）缩放的比例。当mode为hide时默认为0，为show时默认为100 scale：（字符串）要缩放的元素区域，指定为box，则调整边框和内边框的尺寸；指定为content，则调整元素内容的尺寸。指定为both，则同时调整二者尺寸，默认为both
shake 水平或垂直地来回晃动元素	direction：（字符串）up、down、left或right其中之一。如果省略，则默认为left distance：（数字）以像素为单位的晃动距离，默认为20个像素 duration：（数字）每次晃动的速度，默认为140毫秒 mode：（字符串）show、hide、toggle或effect（默认值）其中之一 times：（数字）晃动的次数。如果省略，则默认为3

9

（续）

特效名称的描述	指定特效的选项
size 重新设定元素的宽度和高度。类似于scale，但指定目标尺寸的方式不同	from：（对象）一个对象，其height和width属性指定初始尺寸。如果省略，则从元素当前尺寸开始变化 to：（对象）一个对象，其height和width属性指定最终尺寸。如果省略，则以元素当前尺寸结束变化 origin：（数组）为特效定义基础消失点，指定为包含两个字符串的数组。可设置的值有：top、middle、bottom和left、center、right。默认为['middle','center'] scale：（字符串）元素要缩放的区域，指定为box，就会调整边框和内边框的尺寸；指定为content，就会调整元素内容的尺寸。指定为both，则同时调整二者尺寸，默认为both restore：（布尔）保存和恢复正在执行动画的元素及其子节点的某些CSS属性，并且在应用特效之后对其进行恢复。保存哪些属性是未公开的（包括外边框和内边框的设置），这些属性很大程序上取决于其他选项以及元素所处的环境。只有在某个属性不能按你预期的那样恢复初始值时才要使用这个选项 这个选项默认为false。（注意，scale特效内部使用了size特效，并把restore选项设置为true[①]）
slide 移动元素，使其看起来就像滑入或滑出页面	direction：（字符串）up、down、left或right其中之一。如果省略，则默认值为left distance：（数字）滑动元素的距离。这个值应该小于元素的宽度或高度（按方向而定），并且默认值是元素当前的宽度（direction为left或right时）或者高度（direction为up或down时） mode：（字符串）show（默认值）或hide两者之一
transfer 为临时创建的轮廓元素应用动画，使元素看起像是移动到另一个元素上。必须通过类ui-effects-transfer的CSS规则或指定为选项的类来定义轮廓元素的外观	classname：（字符串）一个将要应用到轮廓元素上的附加类名。指定这个选项不会阻止应用类ui-effects-transfer到元素上 to：（字符串）jQuery选择器，用来指定将轮廓元素移动到的那个目标元素。没有默认值，必须指定这个选项特效才能工作

9.3.2　扩展核心库的动画功能

如果仔细分析在本书中已经讨论过的大部分特效，包括 jQuery UI 提供的列表（参见表 9-1），就可以知道大部分特效都是通过改变元素位置、尺寸以及不透明度来实现的。虽然这给了我们很大的自主权（更不用说 jQuery UI）来创建特效，但是如果能够组合使用对颜色应用动画的能力，那么可创建特效还会大大增加。

核心 jQuery 动画引擎没有对颜色应用动画的能力，因此 jQuery UI 扩展了核心 animate()

① 这个说法不完全正确，scale 特效的确在内部使用了 size 特效，但是只有在 mode 参数不等于 effect 时才设置 restore 为 true。

方法的功能，以便允许对指定颜色值的 CSS 属性应用动画效果。

这个增强的功能支持以下的 CSS 属性：

- ❑ `backgroundColor`
- ❑ `borderBottomColor`
- ❑ `borderLeftColor`
- ❑ `borderRightColor`
- ❑ `borderTopColor`
- ❑ `color`
- ❑ `outlineColor`

因为所有的特效最终都会被这个增强的功能执行，所以特效是如何开始的并不重要——所有指定特效的方式都可以利用这个增强功能。稍后当我们研究 jQuery UI 提供的核心库的其他扩展时，你就会了解这个扩展功能的重要性。

9.3.3 增强的可见性方法

我们在第 5 章中讨论过，如果向核心 jQuery 的基本可见性方法（也就是 `show()`、`hide()` 以及 `toggle()`）提供了持续时间值，这些方法就会使用一个调整宽度、高度以及元素不透明度的预定义特效来显示或隐藏目标元素。但是如果想要更多的选择该怎么办呢？

jQuery UI 通过在核心 jQuery 中扩展这些方法，给予我们这种灵活性以便可以使用表 9-1 中列出的任何特效。这些方法的扩展语法如下所示。

方法语法：扩展的可见性方法

`show(effect,options,speed,callback)`

`hide(effect,options,speed,callback)`

`toggle(effect,options,speed,callback)`

使用指定的特效来显示、隐藏或者切换包装元素的可见性

参数

`effect`	（字符串）调整元素可见性时所使用的特效。可以使用表 9-1 中列出的任何特效
`options`	（对象）为指定的特效提供如表 9-1 所示的选项
`speed`	（字符串|数字）提供 `slow`、`normal`、`fast` 参数之一或者以毫秒为单位的特效持续时间。（可选）如果省略，则默认为 `normal`
`callback`	（函数）可选的回调函数，在特效结束后为每个元素调用。不向此函数传递任何参数，并设置函数上下文为正在执行动画的元素

返回值

包装集

不管是否意识到，你已经看到了一个使用这些扩展的可见性特效的例子。在 jQuery UI 特效实验室，当改变特效下拉列表值的时候，就会使用以下语句删除任何不适合新选中特效的选项控件：

```
$(someSelector).hide('puff');
```

同时，通过下面这句代码来显示适合选中特效的控件：

```
$(someSelector).show('slide');
```

作为高级练习，请复制 jQuery UI 特效实验室，然后将其转变为 jQuery UI 的 Show、Hide 以及 Toggle 实验室：

- 添加一组单选按钮控件来选择这三个可见性方法之一：`show()`、`hide()`以及 `toggle()`；
- 当单击 Apply 按钮的时候，确定选中的是哪个方法，然后执行该方法以取代 `effec()`方法。

可见性方法不是 jQuery UI 通过额外的功能来扩展的唯一一组核心方法。下面来看看扩展的核心方法还有哪些。

9.3.4　为类转换应用动画特效

你可能会回想起，核心 jQuery 的 `animate()`方法可以指定一套 CSS 属性，动画引擎会逐渐修改这些属性，以便创建动画特效。因为 CSS 类是 CSS 属性的集合，所以允许对类转换应用动画也是理所当然的。

jQuery UI 确实也提供了：扩展类转换方法 `addClass()`、`removeClass()`以及 `toggleClass()`能让元素呈现出 CSS 属性变化的动画效果。扩展的方法语法如下所示。

方法语法：扩展的类方法

`addClass(class,speed,easing,callback)`
`removeClass(class,speed,easing,callback)`
`toggleClass(class,force, speed,easing,callback)`

在包装元素上添加、删除或者切换指定的类名。如果省略 speed 参数，这些方法的行为和未扩展的核心方法完全一样

参数

`class`	（字符串）将要添加、删除或者切换的 CSS 类名或者空格分隔的类名列表
`speed`	（字符串|数字）提供 slow、normal、fast 或者以毫秒为单位的特效持续时间。（可选）如果省略，则不会发生动画特效
`easing`	（字符串）将要传入 `animate()`方法的缓动函数的名称。参见第 5 章对 `animate()`的描述以获得更多的信息
`callback`	（函数）在动画结束时调用的回调函数。参见第 5 章对 `animate()`的描述以获得更多的信息
`force`	（布尔）如果指定为 true，则强制 `toggleClass()`方法添加类名，如果指定为 false，则强制删除类名

返回值

包装集

除了扩展这些核心的类转换方法，jQuery UI 还添加了一个新的有用的类操作方法 switchClass()，它的语法如下所示。

方法语法：**switchClass**	
switchClass(removed,added,speed,easing,callback)	
使用一个转换特效来删除指定的一个或者多个类，同时增加指定的一个或者多个特效	
参数	
removed	（字符串）将要删除的 CSS 类名或者空格分隔的类名列表
added	（字符串）将要添加的 CSS 类名或者空格分隔的类名列表
speed	（字符串|数字）（可选地）提供 slow、normal、fast 或以毫秒为单位的特效持续时间。如果省略，则由 animate()方法确定默认值
easing	（字符串）要传入 animate()方法的缓动函数的名称。参见第 5 章对 animate()的描述以获得更多的信息
callback	（函数）在动画结束时调用的回调函数。参见第 5 章对 animate()的描述以获得更多的信息
返回值	
包装集	

jQuery UI 就如何编写代码以动画方式操作元素给了我们很多选择，包括 effct()方法以及对核心的可见性和类转换方法的扩展。

能不能只使用 animate()方法来处理所有这样的情况呢？当然可以。但是考虑到代码的清晰性，使用名为 hide()的方法来隐藏元素（即使是以动画方式）比使用名为 animate()的方法更具有意义。jQuery UI 让我们能够使用对代码上下文最具有意义的方法，而不管是否使用了动画。

jQuery UI 在动画领域提供的另一个扩展是一个相当庞大的缓动函数集合，扩展了 jQuery 核心所提供动画效果。

9.3.5 缓动特效

在第 5 章中讨论动画的时候，我们引入了缓动函数的概念（非正式地称为缓动特效）用来控制动画进行的节奏。jQuery 核心提供了两个缓动特效：linear 和 swing。jQuery UI 将 swing 重命名为 jswing，添加了自己的 swing 版本，并且添加了其他 31 个缓动特效。

我们可以在任何接受散列选项的动画方法中指定一个缓动特效。如前所述，这些选项最终都会被传入 animate()核心方法，所有的动画方法最终都会调用该方法来执行动画或者特效。这些核心选项之一就是 easing，用来识别将要使用的缓动函数的名称。

加载了 jQuery UI 之后，所有可用的缓动特效的清单如下：

- linear
- swing
- easeInQutQuart
- easeInQuint
- easeOutCirc
- easeInOutCirc

- jswing
- easeInQuad
- easeOutQuad
- easeInOutQuad
- easeInCubic
- easeOutCubic
- easeInOutCubic
- easeInQuart
- easeOutQuart
- easeOutQuint
- easeInOutQuint
- easeInSine
- easeOutSine
- easeInOutSine
- easeInExpo
- easeOutExpo
- easeInOutExpo
- easeInCirc
- easeInElastic
- easeOutElastic
- easeInOutElastic
- easeInBack
- easeOutBack
- easeInOutBack
- easeInBounce
- easeOutBounce
- easeInOutBounce

几乎无法用语言来描述每个缓动特效是如何运作的，我们需要在运行中观察它们才能了解指定的缓动特效是如何影响动画的。因此我们可以使用 jQuery UI 缓动实验室页面，观察当把缓动特效应用到不同的动画时它们是如何运作的。该实验室如图 9-6 所示，也可以从文件 chapter9/lab.ui-easings.html 获取。

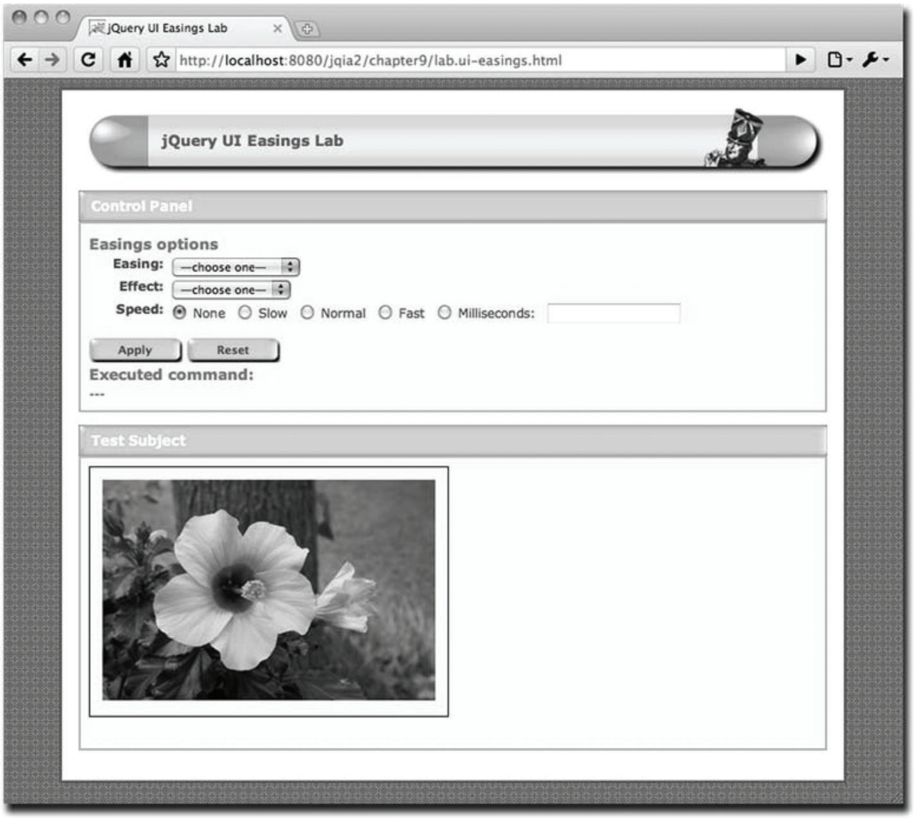


图 9-6　jQuery UI 缓动特效实验室向我们展示了当把不同的缓动特效应用到动画特效时它们是如何运行的

注意　作为补充资源，jQuery UI 在线文档有 SVG 驱动的缓动特效示例，可以从 http://jqueryui.com/demos/effect/#easing 获取。①

这个实验室可以让我们尝试将不同的缓动特效与不同的特效配合使用。为了更好地看到在每个缓动函数运行的变化过程，我们建议尝试如下操作。

(1) 选择要观察的缓动特效。

(2) 选择 `scale` 特效。选中 `scale` 之后，`percent` 选项被硬编码为 25。

(3) 将速度设得非常慢——比 `slow` 的设置还要慢。尝试设置为 10 秒（10 000 毫秒）来观察选中的缓动特效是如何影响测试对象的缩放比例。

下面来看 jQuery UI 提供的另一个实用功能。

9.4　高级定位

CSS 定位能较为轻松地定位页面内的元素。结合使用 jQuery，就能让元素定位变得非常简单前提是知道要在哪里定位元素。

例如，如果我们知道要将元素移动到某个绝对位置，就可以这样写：

```
$('#someElement').css({
  position: 'absolute',
  top: 200,
  left: 200
});
```

但是如果想要相对于另一个元素来定位这个元素，该怎么办呢？例如，将元素放置到另一个元素的右边并顶部对齐，该怎么做？或者将元素放置在另一个元素的下面并中心对齐，该怎么做呢？

没问题，真的。我们可以使用核心 jQuery 方法来获取所涉及的元素的尺寸和位置信息，然后做一些数学处理，再使用处理的结果来对目标元素进行绝对定位。

尽管这不成问题，但实现起来还是需要相当多的代码，如果在计算新位置时不小心搞错公式的话，代码就很容易出问题。这样的代码可读性也不高，尤其是对于那些当初没有参与编写代码的开发者来说，不可能随便看看代码就能知道代码要做什么。

jQuery UI 通过提供一个方法来帮助我们，这个方法不仅抽象出用来计算元素相对于其他元素位置的公式（后面将会看到更多指定元素相对位置的方式），而且可以以一种极其易读的方式来表示。

这个方法是曾在第 3 章中探讨过的 `position()` 方法（用来获取元素相对于其偏移父元素的位置）的重载。语法如下所示。

---

① 这个说法不正确，这个示例是基于 HTML `canvas` 元素的，而不是基于 SVG 的。

方法语法：**position**
position（**options**） 使用 options 提供的信息来对包装元素进行绝对定位 **参数** options　　（对象）提供用来指定如何定位包装集元素的信息，参见表 9-2 **返回值** 包装集

你可以通过提供的实验室页面来观察 jQuery UI 的 position()方法的运行方式。jQuery UI 定位实验室如图 9-7 所示，也可以从 chapter9/lab.ui-positioning.html 获取。

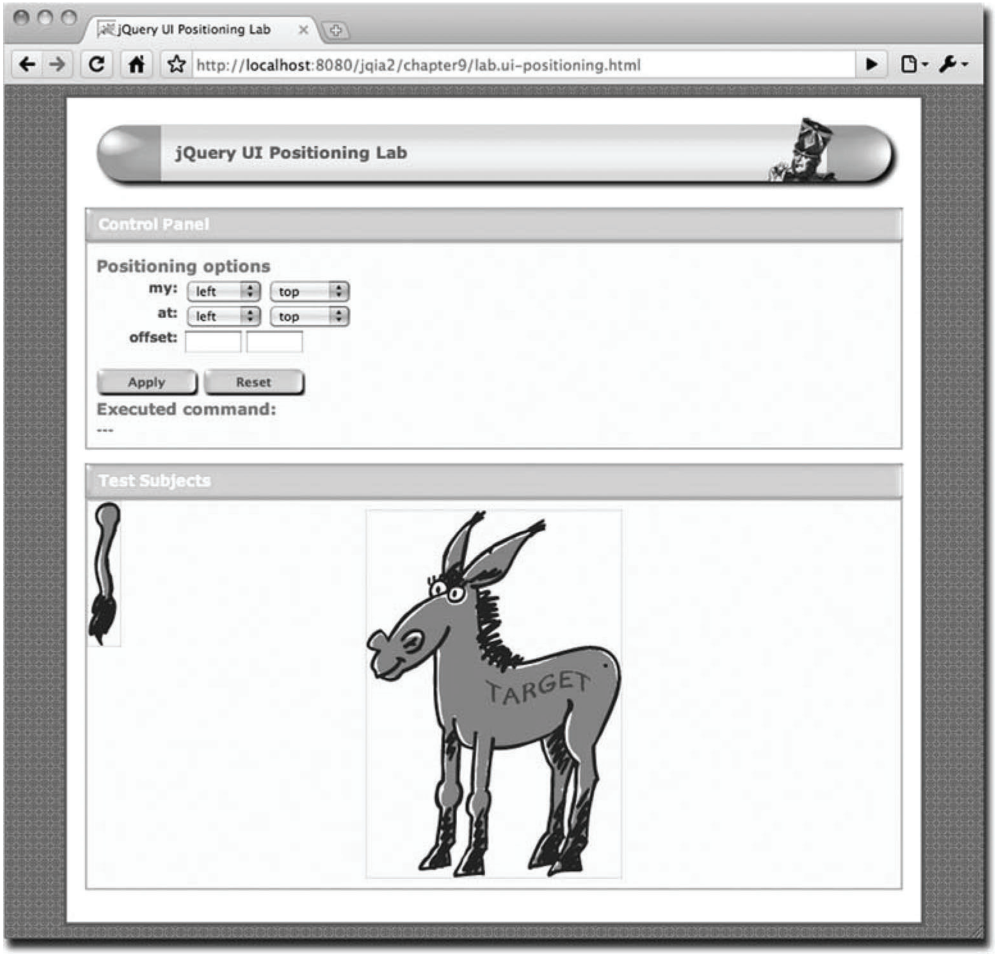


图 9-7　jQuery UI 定位实验室允许我们在运行中观察由 jQuery UI 重载的 position()方法

表9-2　jQuery UI **position()**方法的选项

选　项	描　述	是否在实验室页面中
my	（字符串）指定包装元素（要进行重定位的元素）的位置来使其与目标元素或在at里设定的位置对齐。由top、left、bottom、right以及center中的两个值构成（由一个空格符分隔），其中第一个值是水平坐标值，第二个是垂直坐标值。如果只指定单个值，则另一个默认为center。指定的单个值是被当做是水平的还是垂直的取决于你所使用的值（例如，top是垂直坐标值，而right则是水平坐标值） 示例：top或者bottom right	✓
at	（字符串）指定目标元素的位置来作为重定位元素的目标位置。接受的值和my选项相同 示例：right或者left center	✓
of	（选择器|事件）识别目标元素（作为重定位包装元素的目标）的选择器，或者作为目标位置的包含鼠标坐标的Event实例	✓
offset	（字符串）指定添加到计算出来的位置的任何偏移量，作为两个像素值分别指定left和top。允许负偏移量。例如：10～20 如果指定单个值，它会同时应用于left和top。默认值是0	✓
collision	（字符串）当被定位的元素在任何方向超出窗体边界时，用来指定应用的规则。接受下面所示的值（先水平后垂直规则）。 flip：默认值，翻转元素到相反的方向，然后再次运行冲突检测。如果两个方向均不在窗体内，则回退到center位置 fit：保持元素在原方向，通过调整位置以便被定为的元素不会超出窗体边界 none：禁用冲突检测 如果指定单个值，它会同时应用到水平和垂直方向	
using	（函数）函数，用来替换内部函数来改变元素的位置。为每个包装元素调用，传递由left和top属性组成的散列对象（也就是计算出的目标位置）的单个参数，需要定位的元素被设置为函数上下文	

在阅读表9-2中的选项时，请使用定位实验室来熟悉选项的操作。如果你一次就能把尾巴钉在合适的位置上，那就给自己点奖励吧。

你可能一边观察这些选项的名称一边对自己说，“他们在想什么？at？my？of？这些到底是什么？”

一开始你可能不明白这些名字的含义。但如果你在使用定位实验室的时候观察生成的语句，就会发现它们的玄机。如果还没有明白，则思考下面的语句：

```
$('#someElement').position({
  my: 'top center',
  at: 'bottom right',
  of: '#someOtherElement'
});
```

这看起来就像是一句英文！即使是没有读过计算机代码的人都能差不多知道这个语句要做什

么。（为什么计算机的硬件设备要用到这么别扭的符号呢？）

大部分 API 都可以从这种“疯狂”中受益。

9.5　小结

在本章中我们开始对 jQuery UI 进行深入的研究，在本书结束前我们会一直探讨 jQuery UI 这个话题。

我们了解到作为核心 jQuery 库的官方伙伴，jQuery UI 享有特殊的地位，也知道了如何从 http://jqueryui.com/download 下载该库的自定义版本（同时包括一个预定义主题）。还看到了下载压缩包中的内容，以及将该库添加到 Web 应用程序目录结构的典型方法。

然后，我们探讨了 jQuery UI 库的主题设计能力，以及如何布局其定义的 CSS 类，包括如何通过命名约定来组织这些 CSS 类。

我们考察了官方的 ThemeRoller 在线应用程序（http://jqueryui.com/themeroller/），可以用来调整一个预定义的主题或者完全从头创建新的主题。

本章的剩余部分考察了 jQuery 为核心库所做的扩展。

我们看到了核心动画引擎是如何被扩展成提供大量已命名的特效，可以很容易地使用 `effect()`方法来启动这些特效。

我们也看到 jQuery UI 如何扩展可见性方法 `show()`、`hide()`和 `toggle()`来和这些新特效共同工作。相同的扩展方式也被用于类转换方法：`addClass()`、`removeClass()`、`toggleClass()`以及新定义的 `switchClass()`方法。

然后我们讨论了 30 个 jQuery UI 添加的缓动函数，动画引擎用它们来控制动画效果。

最后，我们考察了对核心 `position()`方法的一个扩展，它允许我们以非常易读的方式根据相对于其他元素的位置或者鼠标事件的位置来定位元素。

这只是个开始。继续阅读下一章，我们将会学习 jQuery UI 的另一个主要部分：鼠标交互。

第 10 章

jQuery UI 鼠标交互：跟随鼠标的移动

10

本章内容

- 核心鼠标交互部件
- 实现拖放操作
- 使元素可排序
- 允许改变元素尺寸
- 使元素可选择

大多数可用性专家认为：直接操作是产生良好用户界面的关键。与模拟用户活动的界面相比，允许用户直接操作元素并立即看到操作所产生的效果是一个更好的用户体验。

以对列表中的元素排序为例。如何让用户为元素指定一种排列顺序呢？如果只使用 HTML 4 提供的一套基本控件，就没有足够的灵活性来完成这件事情。在元素列表后面加个文本框（用户必须在其中输入序号值），可用性又大打折扣。

如果能够让用户选中并拖动列表中的元素，把它们来回移动，就能排出满意的结果呢？这种机制（使用直接操作）显然是上乘的做法，但是只使用 HTML 基本控件集是做不到的。

核心交互部件（关注直接操作）是创建 jQuery UI 的基石，它们在呈现给用户的用户界面类型方面提供了更强大的功能和更高的灵活性。

核心交互部件为在页面使用鼠标指针添加了高级的行为。我们可以自己使用这些交互部件（贯穿本章始终），同时它们也是创建许多其他的 jQuery UI 库的基石。

核心交互部件包含如下内容。

- 拖动——在页面上移动元素（10.1 节）。
- 放置——把拖动的元素放置到其他元素上（10.2 节）。
- 排序——使元素按顺序排列（10.3 节）。
- 改变尺寸——改变元素的尺寸（10.4 节）。
- 选择——使通常情况下不可选择的元素转变为可选择的元素（10.5 节）。

在学习本章的过程中你将看到，核心交互部件是互为基础的。为了发挥本章的最大作用，建

议你按照顺序学习本章。这可能是篇幅很长的一章，但是 jQuery UI 方法有着某种程度的一致性（反映在本章各节的结构中），一旦你熟悉了这些方法是如何组织的之后，就能很容易地学习本章全部的知识。

与鼠标指针交互是任何 GUI 不可或缺和核心的部分。尽管有一些简单的鼠标指针交互动作被内置在 Web 界面中（例如单击），但 Web 本身并不支持某些可用于桌面应用的高级交互方式。这个缺点的最好例证是缺少对拖放操作的支持。

拖放操作是桌面用户界面中普遍存在的一种交互技术。例如，在任何桌面系统的 GUI 文件管理器中，我们都可以轻松地在文件系统中将文件在文件夹之间来回拖放，实现复制或者移动操作，甚至把这些文件拖放到垃圾桶或者废纸篓图标上来删除它们。但是尽管这种交互风格在桌面应用中非常流行，在 Web 应用中却很少见到，这主要是因为现代浏览器缺少对拖放操作的原生支持，而且正确地实现这个功能也是一个相当艰巨的任务。

"艰巨？"你可能会嘲笑道，"这只需要一点鼠标事件和 CSS 小技巧，有什么大不了的？"

尽管高层次的概念并不难以把握（特别是在使用强大的 jQuery 的情况下），但事实证明实现拖放支持的细节（特别是以一种健壮且独立于浏览器的方式），会很快变得痛苦起来。不过就像之前 jQuery 和它的部件为我们减轻负担那样，jQuery UI 再次以相同的方式添加了对拖放操作的直接支持。

但是在能够拖动和放置元素之前，先来学习如何拖动元素，这也是我们的起点。

10.1　来回拖动元素

尽管在大部分的词典中找到可拖动（draggable）这个词很困难，但是这个术语通常用来指拖放操作中可以被来回拖动的项目。同样，它也是 jQuery UI 用来描述这类元素的术语，并把它用作方法名称。这些方法可以让匹配集的元素也具备可拖动功能。

在介绍 `draggable()` 方法的语法之前，先来花点时间讨论一下 jQuery UI 中经常用到的习惯用法。

为了尽可能地减少对方法命名空间的侵占，jQuery 中的很多方法都是根据传入参数的性质来服务于多重目的。这不是什么特别新的概念——我们已经在核心 jQuery 中看到过很多类似的情况。但是 jQuery UI 将方法重载带入了更高层级。我们将看到相同的方法可以在很多关联的动作上使用。

`draggable()` 方法就是一个绝好的例子。这个方法不仅可以用来使元素可拖动，而且用来控制元素各个方面的可拖动性质（包括禁用、销毁以及重新启用元素的可拖动性），同时还用来设置和获取单个的可拖动性选项。

因为所有这些操作的方法名称都是一样的，所以我们只能通过参数列表来区分将要进行的操作。通常，这种差异使用传入的第一个字符串参数作为判定器，来识别要执行的操作。

例如，要禁用可拖动元素的可拖动性质，可以编写如下代码：

```
$('.disableMe').draggable('disable');
```

注意 如果你已经使用较早版本的 jQuery UI，那可能还记得当时是通过定义不同的方法（例如 `draggableDisable()`和 `draggableDestroy()`）来提供各种不同操作的。这些方法已经不存在了，它们被更加简明的多用途方法（例如 `draggable()`）替代了。

`draggable()`方法各种形式的语法如下所示。

命令语法：draggable

`draggable(options)`

`draggable('disable')`

`draggable('enable')`

`draggable('destroy')`

`draggable('option', optionName, value)`

根据指定的选项使包装集中的元素可拖动，或者基于作为第一个参数传入的操作字符串来执行其他的可拖动性操作

参数

`options`	（对象）散列对象，由要应用到包装集元素上的选项（参见表 10-1 所述）组成，从而使得这些元素可拖动。如果省略（并且未指定任何其他参数）或为空，则元素将可以在窗体内任意拖动
`'disable'`	（字符串）暂时禁用包装集中任何可拖动元素的可拖动性。并不删除元素的可拖动性，可以通过调用这个方法的`'enable'`变体来恢复元素的可拖动性
`'enable'`	（字符串）重新启用包装集中任何可拖动元素（它们的可拖动性已经被禁用了）的可拖动性。注意这个方法不会向任何不可拖动的元素添加可拖动性
`'destory'`	（字符串）删除包装集中元素的可拖动性
`'option'`	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（这个元素必须是可拖动的元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 `optionName` 参数
`optionName`	（字符串）要设置或返回的选项名称的值（参见表 10-1）。如果提供 `value` 参数，则这个值就成为选项的值。如果没有提供 `value` 参数，则返回已命名选项的值
`value`	（对象）要设置的选项的值（通过 `optionName` 参数标识）

返回值

包装集，除了返回选项值的情况

将以上这么多方法的变体都封装到一个方法中。下面首先考察如何使元素可拖动，逐步深入研究这个部件。

10.1.1　使元素可拖动

查看列表中 `draggable()`方法的变体，我们可能会期望通过调用 `draggable('enable')` 方法来使得包装集中的元素可拖动，但这绝对是大错特错！

为了使得元素可拖动，需要使用带有对象组成的参数（对象的属性指定了可拖动性选项，参见表 10-1）或者不带任何参数（使用所有的默认设置）调用 `draggable()`方法。我们马上就会看到 `enable` 调用是用来做什么的。

当一个元素成为可拖动元素时，类 `ui-draggable` 就会被添加到这个元素上。这不仅有助于找出可拖动的元素，而且可以作为使用 CSS 应用视觉样式的标识物（应该选择这么做[①]）。也可以标识正在被拖动的元素，因为在拖动操作过程中，会向正在被拖动的元素添加类 `ui-draggable-dragging`。

可拖动性选项有很多，因此为了帮助熟悉这些选项，我们提供了一个 jQuery UI 可拖动实验室页面。在学习本节其余部分的时候可以在浏览器中打开 chapter10/draggables/lab.draggbles.html 页面。这个页面的显示如图 10-1 所示。

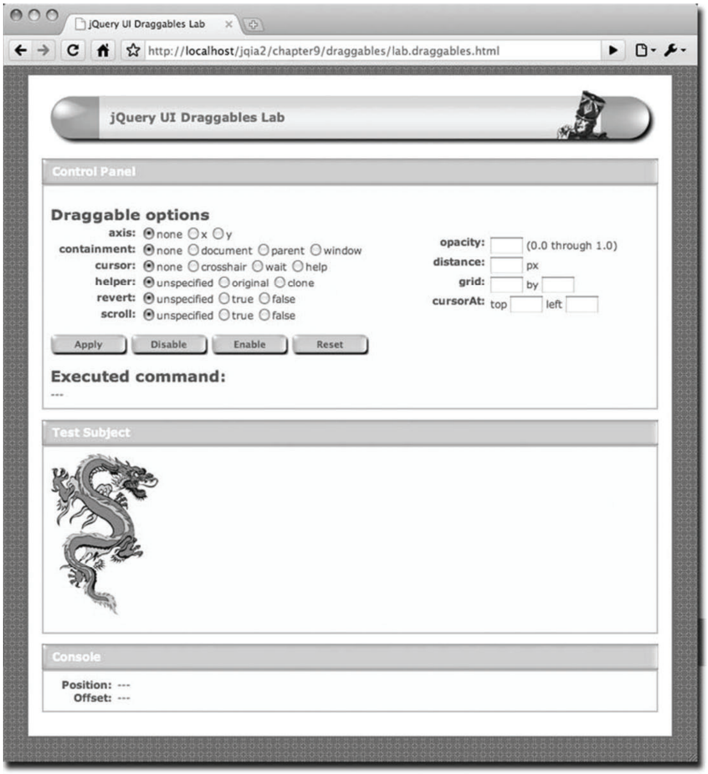


图 10-1　可拖动实验室页面会帮助我们熟悉 jQuery UI 中可拖动部件的很多选项

---

① 通过 jQuery UI 部件添加到元素上的类名来向元素应用额外的样式规则是推荐的做法。

draggable()方法的各种选项提高了灵活性，让我们可以精确地控制拖动操作是如何发生的。这些选项的描述如表 10-1 所述。可拖动实验室页面中展现的选项在表 10-1 中通过“是否在实验室中”列来识别。在学习过程中请务必尝试这些选项的各种效果。

表10-1　jQuery UI的**draggable()**方法的选项

选　　项	描　　述	是否在实验室页面中
addClasses	（布尔）如果指定为false，则阻止向可拖动元素添加类ui-draggable。如果不需要添加此类并且页面上有很多需要添加可拖动性的元素，出于性能原因我们可能会将该选项设置为false 尽管这个选项是用复数表示的名称，但是它不会阻止向元素添加其他类，例如不会阻止在拖动操作时向元素添加的ui-draggable-dragging类	
appendTo	（元素|选择器）当创建助手（helper）时（参见下面的helper选项），指定将助手追加到的DOM元素。如果未指定，则任何助手都会被追加到可拖动元素的父元素中	
axis	（字符串）如果指定为x或y，则限制拖动操作在指定的坐标轴上移动。例如，指定x只允许元素在水平坐标轴上移动。如果未指定或者指定为任何其他值，则不限制移动的坐标轴	✓
cancel	（选择器）指定一个选择器来标识不允许发生拖动操作的元素。如果未指定，则使用选择器:input,option。 注意它不会阻止这些元素成为可拖动的元素，它只会阻止这些元素发生拖动操作。这些元素依然被认为是可拖动的并且会被添加ui-draggable类	
connectToSortable	（字符串）标识可排序列表，以便将可拖动元素放置到列表中从而使其成为列表的一部分。如果指定此选项，那么也应该指定helper选项为clone 这个选项需要可排序部件的支持，我们会在10.3节介绍这个部件	
containment	（元素|选择器|数组|字符串）定义限制拖动操作的区域。如果未指定或者指定为document，则在文档内移动范围是没有限制的 字符串window会把移动限制在可见的视窗内，而字符串parent会把移动限制在元素的直接父元素内 如果指定为一个元素，或者用来识别一个元素的选择器，则移动会被限制在这个元素内 也可以将相对于文档的任意矩形区域指定为包含4个数字的数组，按照如下格式来指定矩形的左上角和右下角坐标：[x1,y1,x2,y2]	✓
cursor	（字符串）在拖动操作中使用的鼠标指针光标的CSS名称。如果未指定，则默认为auto	✓
cursorAt	（对象）指定光标在被拖动元素内的相对位置（在拖动操作中）。可以指定为包含left或right属性的对象，或者指定为包含top或bottom属性的对象 例如：在拖动过程中，cursorAt:{top:5,left:5}会把光标放置在距离元素左上角5个像素的位置 如果未指定，光标会停留在单击元素的位置	✓

（续）

选　　项	描　　述	是否在实验室页面中
delay	（数字）在按下鼠标事件之后，开始拖动操作之前延迟的毫秒数。这可以防止意外的拖动，只有在用户保持鼠标按钮处于按下状态一定时间后才执行拖动操作 默认值为0，表示没有定义延迟	
distance	（数字）在开始拖动操作之前必须拖动的以像素为单位的距离。这也可以被用来防止意外的拖动。如果未指定，则距离默认为1个像素	✓
drag	（函数）指定在可拖动元素上创建的函数，用来作为drag事件的事件处理器。有关这个事件的更多细节，请参见表10-2	
grid	（数组）包含两个数字的数组，分别用来指定（在一个拖动操作中）拖动操作将会被“吸附”到的水平和垂直距离。网格的原点是被拖动元素的初始位置 如果未指定，则不定义网格	✓
handle	（元素|选择器）指定元素或选择元素的选择器来作为拖动操作的触发器。为了确保正确工作，此元素必须是可拖动元素的子元素 当指定此选项时，只有单击handle指定的元素才会发生拖动操作。默认情况下，单击可拖动元素内的任意位置都会发生拖动操作	
helper	（字符串|函数）如果未指定或者指定为original，则在拖动操作中移动可拖动的元素。如果指定为clone，则创建一个可拖动项目的副本，在拖动操作中移动这个副本 可以指定一个函数，用来允许我们创建并返回一个作为拖动助手的新DOM元素	✓
iframeFix	（布尔|选择器）通过阻止<iframe>元素捕捉鼠标移动事件来防止此元素干涉拖动操作。如果指定为true，则在拖动操作中会阻止所有iframe元素的鼠标移动事件。如果指定为选择器，则会阻止该选择器标识的所有iframe元素的鼠标移动事件	
opacity	（数字）一个0.0 ~ 1.0之间的数字，指定被拖动元素或者助手的不透明度。如果省略，则在拖动过程中不改变元素的不透明度	✓
refreshPositions	（布尔）如果指定为true，则会在拖动过程中的每次鼠标移动事件时重新计算所有可拖动元素的位置（详见10.2节）。因为它的性能开销很大，所以最好只在高度动态的网页中使用它	
revert	（布尔|字符串）如果指定为true，则被拖动元素会在拖动操作结束时恢复到它的初始位置。如果使用字符串invalid，则元素只有在没有被放入到可放置的元素上时才恢复到初始位置；如果指定为valid，则元素只有在被放入到可放置的元素上时才恢复到初始位置 如果省略或指定为false，则被拖动元素不会恢复到它的初始位置	✓
revertDuration	（数字）如果revert为true，则指定被拖动元素恢复到它的初始位置的毫秒数。如果省略，则使用默认值500	
scope	（字符串）用来将可拖动元素和可放置元素关联起来。具有和可放置元素相同名称范围的可拖动元素将自动被可放置元素所接受。如果未指定，则使用默认范围default （当讨论可放置部件时这个选项会更有意义。）	

（续）

选 项	描 述	是否在实验室页面中
scroll	（布尔）如果设置为false，则在拖动操作中阻止容器自动滚动。如果省略或指定为true，则启用自动滚动	✓
scrollSensitivity	（数字）表示光标指针距离视窗边界的距离（以像素为单位）为多少时发生自动滚屏。如果省略，则默认为20像素	
scrollSpeed	（数字）自动滚动的滚动速度。默认值时20。较低该值用来减慢滚动速度，较高的值用来加快滚动速度	
snap	（选择器|布尔）指定用来标识页面上目标元素的选择器，每当被拖动的元素接近这些目标元素时，被拖动的元素就会“吸附”到目标元素的边界。指定为true是选择器.ui-draggable的简写形式，这会使所有其他的可拖动元素成为目标元素	
snapMode	（字符串）指定目标元素的哪个边界可以成为被拖动元素的吸附对象。字符串outer指定只有目标元素的外部边界才能成为吸附对象，而inner指定只有目标元素的内部边界才能成为吸附对象。字符串both（默认值）指定目标元素的内外边界都能成为吸附对象	
snapTolerance	（数字）如果启用snap，以像素为单位指定发生吸附时元素到目标元素边界的距离。默认值是20像素	
Stack①	（对象）一个散列对象，用来控制拖动操作中分组元素的z-index叠放。每当拖动元素时，这个元素就成为了分组中所有其他可拖动元素中层级最高的元素（通过设置z-index的值）。 也可以通过stack对象的min属性来指定最小值，z-index不会低于这个最小值	
start	（函数）指定在可拖动元素上创建的函数，用来作为dragstart事件的事件处理器。有关这个事件的更多细节，请参见表10-2	
stop	（函数）指定在可拖动元素上创建的函数，用来作为dragstop事件的事件处理器。有关这个事件的更多细节，请参见表10-2	
zIndex	（数字）指定拖动操作中可拖动元素的z-index值。如果省略，则在拖动操作中可拖动元素的z-index值不会改变	

所有这些选项为如何进行拖动操作提供了很大的灵活性。但是还没有结束。可拖动部件还能灵活地控制页面中的其他元素。下面就来看看它是如何完成的。

10.1.2 可拖动性事件

在表 10-1 中，可以看到有三个选项允许我们在可拖动元素上注册事件处理器：drag、start 和 stop。这些选项是为三个自定义事件绑定事件处理器的便捷方法，jQuery 会在拖动操作的各个阶段触发这些事件：dragstart、drag 和 dragstop（参见表 10-2）。这个表格（以及下面所有描述事件的表格）显示了可绑定自定义事件的名称、用来指定处理器函数的选项名称，以及对事件的描述。

10

① 这里对 stack 的解释已经过时了，在 jQuery UI 1.8.0 及其以上版本中，stack 是一个选择器而非散列对象，并且 min 也变成了局部变量。

表10-2　为可拖动部件触发的jQuery UI事件

事　件	选　项	描　述
dragstart	start	当拖动操作发生时触发
drag	drag	在拖动操作中为鼠标移动事件连续触发
dragstop	stop	当拖动操作终止时触发

可以在可拖动元素祖先层次中的任何元素上创建这些事件的处理器，以便在这些事件发生时接收通知。例如，我们可能希望以某种全局的方式来响应 dragstart 事件，可以在文档的主体（body）上创建该事件的处理器：

```
$('body').bind('dragstart',function(event,info){
  say('What a drag!');
});
```

不管处理器是在哪里创建的，无论是通过选项还是 bind()方法创建的，都会向处理器传入两个参数：一个鼠标事件实例，一个包含拖动事件当前状态信息的对象。后者的属性如下所示。

- helper——包含正在被拖动元素的包装集（要么是原始元素，要么是原始元素的副本）。
- position——一个对象，其 top 和 left 属性给出了被拖动元素相对于偏移父元素的位置。对于 dragstart 事件来说该属性可能是 undefined。
- offset——一个对象，其 top 和 left 属性给出了被拖动元素相对于文档页面的位置。对于 dragstart 事件来说该属性可能是 undefined。

可拖动实验室页面创建了可拖动事件处理器，并且使用传入处理器的信息在 Console 面板中显示被拖动元素的位置。

一旦使用 draggable()方法的第一种形式（传入 options 对象）来让元素成为可拖动元素，就可以使用其他形式来控制元素的可拖动性。

10.1.3　控制可拖动性

我们在前一节介绍过，可以调用带有选项散列值的 draggable()方法来为包装元素建立可拖动性。在元素成为可拖动元素后，我们有时可能想要暂时禁用元素的可拖动性，但又不想丢失创建此特效时的所有选项。

可以通过调用 draggable()方法的如下形式来暂时禁用元素的可拖动性：

```
$('.ui-draggable').draggable('disable');
```

包装集中任何可拖动的元素都将暂时变成不可拖动元素。在上一个示例中，我们禁用了页面上所有可拖动元素的可拖动性。

为了恢复这类元素的可拖动性，可以使用如下语句：

```
$('.ui-draggable').draggable('enable');
```

这将重新启用任何已经被禁用的可拖动元素的可拖动性。

警告　如前所述，不可以使用 draggable('enable')方法来向不可拖动元素应用可拖动性。这个方法的 enable 形式只能用来重新启用之前被禁用的可拖动元素的可拖动性。

如果想要永久禁用可拖动的元素可拖动性，将它们恢复到可拖动之前的状态，那么可以使用下面这个语句：

```
$('.ui-draggable').draggable('destroy');
```

这个方法的 destory 变体会从元素上删除所有残余的可拖动性。

具有多种用途的 draggable()方法的最后一种格式允许我们在可拖动元素的整个生命周期内设置或者获取单个选项的值。

例如，在可拖动元素上设置 revert 选项，可以使用下面这行代码：

```
$('.whatever').draggable('option','revert',true);
```

这将会设置包装集中第一个元素（这个元素必须是可拖动的）的 revert 选项为 true[①]。若在不可拖动元素上设置选项，则不会产生任何作用。

如果希望获取可拖动元素的选项，那么可以这么写：

```
var value = $('.ui-draggable').draggable('option','revert');
```

这将会获取包装集中第一个元素 revert 选项的值，前提是这个元素必须是可拖动的，否则只能得到 undefined。

在屏幕上到处拖动元素是很好，但是这真的有用吗？它的乐趣或许会持续一段时间，但就像玩溜溜球一样，只能风靡一时（除非我们是狂热爱好者）。在实际应用中，我们可能使用它来让用户在屏幕上移动模块化的元素（如果做得好的话，可以在 cookies 或者其他的持久化机制中记录用户选择的位置），或者用在游戏、猜谜游戏中。但是拖动操作真正的闪光点出现在放置这些拖动的元素时发生的有趣事情上。因此下面就来看看如何将可放置元素与可拖动元素配合使用。

10.2　放置可拖动元素

与可拖动元素相对应的是*可放置元素*——即可以接受拖动元素的元素，并在接受那些可拖动元素时完成一些有趣的事情。从页面元素创建可放置项目和创建可拖动元素类似。事实上还会更加简单，因为要考虑的选项更少。

和 draggable()方法类似，droppable()方法也有几种形式：一个形式用来初始化创建可放置的元素，以及其他的用来影响创建后的可放置元素。它的语法如下所示。

① 原文描述不严谨（容易让人误认为只更新第一个元素的选项），事实上通过'option'方法来设置选项时，会同时更新包装集中所有元素的这个选项。为了验证此说法，请在可拖动实验室页面执行如下代码：

```
$('.module').draggable();
$('.module').draggable('option','revert', true);
```

然后拖动页面中 Control Panel、Test Subject 和 Console 3 个面板，你会发现放开鼠标后 3 个面板都能回到其原始位置。

命令语法：**droppable**	
droppable(options)	
droppable('disable')	
droppable('enable')	
droppable('destroy')	
droppable('option', optionName, value)	
根据指定的选项使包装集中的元素可放置，或者基于作为第一个参数传入的操作字符串来执行其他的可放置性操作	
参数	
options	（对象）一个散列对象，由要应用到包装集元素上的选项组成（参见表 10-3），从而使得这些元素可放置
'disable'	（字符串）暂时地禁用包装集中任何可放置元素的可放置性。并不删除元素的可放置性，可以通过调用这个方法的'enable'变体来恢复元素的可放置性
'enable'	（字符串）重新启用包装集中任何可放置元素（它们的可放置性已被禁用）的可放置性。注意这个方法不会向任何不可放置的元素添加可放置性
'destroy'	（字符串）删除包装集中元素的可放置性
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是可放置元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
optionName	（字符串）要设置或者返回的选项名称的值（参见表 10-3）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
value	（对象）要设置的选项的值（通过 optionName 参数来标识）
返回值	
包装集，除了返回选项值的情况	

下面来看看如何才能使元素成为可放置的元素。

10.2.1　使元素可放置

要使元素成为可放置的元素，只需要将它们收集到包装集，调用带散列对象选项的 `droppable()`方法（或者不带任何参数，以便使用选项的默认值）。当元素成为可放置元素后，会向其添加 `ui-droppable` 类。这与使元素可拖动的方式类似，但是需要的选项更少，如表 10-3 所示。

与可拖动元素一样，我们也提供了一个 jQuery UI 可放置元素实验室页面（如图 10-2 所示），用来实时演示大部分可放置选项。



图 10-2 可放置实验室页面允许我们实时观察可放置元素的选项

在 chapter10/droppables/lab.droppables.html 找到这个页面，并在浏览器中加载它。当阅读表 10-3 的选项描述时，可以使用这个页面来操作可放置元素的选项。

尽管可放置元素与可拖动元素相比，其选项较少，但是很明显，有更多的事件和状态与可放置元素相关。下面就来详细考察这些状态和事件。

10

表10-3　jQuery UI的**droppable()**方法的选项

选　　项	描　　述	是否在实验室页面中
accept	（选择器 | 函数）指定标识可拖动元素（这些元素将被放置元素所接受）的选择器，或者用来筛选页面上所有可拖动元素的函数。为所有可拖动的元素调用此函数，并将该元素引用作为第一个参数传入此函数。从函数返回true，表示接受这个可拖动元素为放置元素 如果省略，则接受所有的可拖动元素	✓
activate	（函数）指定在可放置元素上创建的函数，用来作为dropactivate事件的事件处理器，这个事件会在放置操作开始时触发。有关这个事件的更多细节，请参见表10-4	✓
activeClass	（字符串）每当进行带有可接受元素的拖动操作时，将一个或者多个类添加到可放置元素上。可以指定多个的类名，使用空格符来分割它们 如果省略，则在可接受的拖动操作中不会添加类到可放置的元素上	✓
addClasses	（布尔）如果指定为false，则阻止向可放置元素添加类ui-droppable。如果不需要添加此类并且页面上有很多需要添加可放置性的元素，出于性能原因，我们可能会将选项设置为false	
deactivate	（函数）指定在可放置元素上创建的函数，用来作为dropdeactivate事件的事件处理器，这个事件会在放置操作终止时触发。有关这个事件的更多细节，请参见表10-4	✓
drop	（函数）指定在可放置元素上创建的函数，用来作为drop事件的事件处理器。有关这个事件的更多细节，请参见表10-4	✓
greedy	（布尔）可放置性事件通常会传播到嵌套的可放置元素上。如果设置这个选项为true，则会阻止事件传播	
hoverClass	（字符串）每当可接受的可拖动元素在可放置元素上悬停时，要添加到可放置元素的一个或多个类名。可以指定多个类名，使用空格符来分割它们。 如果省略，则在可接受的悬停操作中不会添加类到可放置的元素上	✓
out	（函数）指定在可放置元素上创建的函数，用来作为dropout事件的事件处理器。有关这个事件的更多细节，请参见表10-4	✓
over	（函数）指定在可放置元素上创建的函数，用来作为dropover事件的事件处理器。有关这个事件的更多细节，请参见表10-4	✓
scope	（字符串）用来将可拖动元素和可放置元素关联起来。具有和可放置元素相同名称范围的可拖动元素将自动被可放置元素所接受。如果未指定，则使用默认范围default	
tolerance	（字符串）控制在哪种情况下被拖动元素是悬停到可放置元素上的。它的值如下所示： ❑ fit——可拖动元素必须完全位于可放置元素内 ❑ pointer——鼠标光标必须进入可放置元素内 ❑ touch——可拖动元素的任意部分必须和可放置元素重叠 ❑ intersect——至少有50%的可拖动元素必须和可放置元素重叠 ❑ 如果省略，则默认使用intersect	✓

10.2.2　可放置性事件

跟踪拖动事件的状态非常简单：元素要么处于被拖动状态，要么处于非拖动状态。但是当引入可放置元素时，事件就会变得有点复杂。不仅需要考虑可拖动元素，而且需要考虑可拖动元素

与接受它的可放置元素之间的交互。

一张图片胜过千言万语，图 10-3 描述了在拖放操作中导致操作转换的状态和事件。

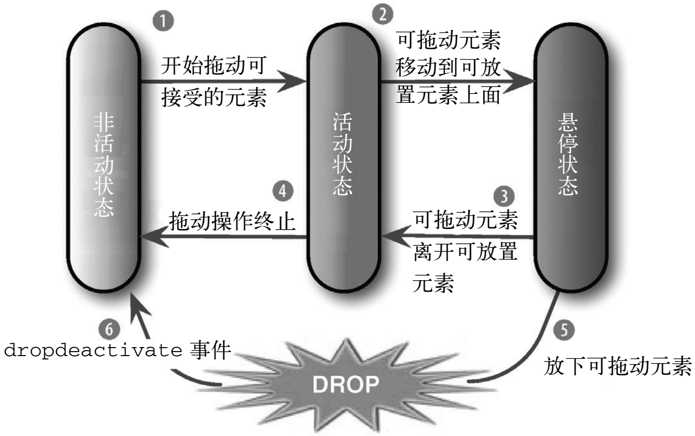


图 10-3 可放置元素经历的状态和转变取决于活动的可拖动元素与可放置元素在拖放操作中的交互

一旦创建了可放置元素，这个元素就将处于*非活动状态*——它已经准备好接受可拖动元素了，但是由于没有执行拖动操作，因此所有一切都在静静地等待。一旦开始拖动操作，事情就变得有趣起来。

❶ 当开始拖动某个能被可放置元素所接受的可拖动元素时（查阅 accept 和 scope 选项来了解哪些元素是*可接受的*元素），就会触发一个 dropactivate 事件且可放置元素处于活动状态。

为 dropactivate 创建的任何处理器都将依照一般的事件传播规则触发，除非指定 greedy 选项为 true，在这种情况下只调用可放置元素上的处理器。

此时，将对可放置元素应用任何由 activeClass 提供的类名。

❷ 如果被拖动的元素移动到某个位置，这个位置满足可放置元素的悬停条件（由 tolerance 选项指定），就会触发 dropover 事件（将调用相应的处理器），并且可放置元素就会进入悬停状态。

任何由 hoverClass 选项提供的类名正是在这个时候被应用到了可放置元素上。

这时有两个可能的转变：释放鼠标按钮来终止拖动操作，或者继续移动可拖动元素。

❸ 如果可拖动元素被移动到的位置不再满足可放置元素的悬停条件（依然是通过 tolerance 规则来判定），就会触发 dropout 事件，并且从可放置元素上删除任何由 hoverClass 指定的类名。

此时可放置元素恢复到活动状态。

❹ 如果拖动操作在活动状态时终止，就会触发 dropdeactivate 事件，并从可放置元素上删除任何由 activeClass 指定的类名，可放置元素恢复到非活动状态。

10

❺ 然而，如果拖动操作在悬停状态时终止，则可拖动元素就被认为是放置到了可放置元素上，就触发两个事件：drop 和 dropdeactivate。

❻ 可放置元素恢复到非活动状态，并从可放置元素上删除任何由 activeClass 指定的类名。

注意　“放置”不是一个状态，而是瞬间完成的事件。

表 10-4 概括了可放置元素的事件。

表10-4　为可放置部件触发的jQuery UI事件

事　件	选　项	描　述
dropactivate	activate	当开始拖动被可放置元素所接受的可拖动元素时触发
dropdeactivate	deactivate	正在进行的放置操作终止时触发
dropover	over	当可接受的可拖动元素悬停在可放置元素上方（由可放置元素的 tolerance选项所定义）时触发
dropout	out	当可拖动元素移出可放置元素区域时触发
drop	drop	当拖动操作（在可拖动元素悬停在可放置元素的时候）终止时触发

所有的可放置事件处理器都接收两个参数：一个鼠标事件实例，以及一个包含拖放操作当前状态信息的对象。这个对象的属性如下所示。

- helper——包含正在被拖动的助手元素的包装集（要么是原始元素要么是原始元素的副本）。
- draggable——包含当前可拖动元素的包装集。
- position——一个对象，其 top 和 left 属性给出了被拖动元素相对于偏移父元素的位置。对于 dragstart 事件它可能是 undefined。
- offset——一个对象，其 top 和 left 属性给出了被拖动元素相对于文档页面的位置。对于 dragstart 事件它可能是 undefined。

可以使用可放置元素实验室来确保理解这些事件和状态转换。和其他实验室一样，Control Panel 允许我们指定在单击 Apply 按钮后要应用到可放置元素上的选项。Disable 和 Enable 按钮分别用来禁用和启用可放置元素（使用 droppable()方法相应的变体），Reset 按钮用来将表单恢复到其初始状态，并销毁实验室页面中放置目标元素的可放置能力。

Test Subjects 面板有六个可拖动元素，以及一个在单击 Apply 按钮后会成为可放置目标的元素（我们称为 Drop Zone）。在 Drop Zone 下面是灰色的文本元素，写着 Activate、Over、Out、Drop 以及 Deactivate。当触发相应的可放置事件时，对应的文本元素（我们称之为事件指示器）会暂时高亮来指示被触发的事件。（你明白实验室是如何实现这个效果的吗？）

下面就使用这个实验室来深入探索可放置部件。

- **练习 1**——在这个练习中，先从熟悉 accept 选项开始，这个选项用来说明可放置元素可接受的可拖放元素都有哪些。尽管可以将这个选项设置为任意的 jQuery 选择器（甚至是一个函数，来通过编程做出合适的决定），但是出于实验室的目的，我们将关注那些拥有特

定类名的元素。具体说来，我们可以指定包含 flower、dog、motorcycle 和 water 中任意类名的选择器，只需要选中 accept 选项中相应的复选框控件即可。

Test Subjects 面板左侧有 6 个可拖动图片元素，每个元素都分配了一个或两个类名（基于图片显示的内容）。例如，左上角的可拖动元素拥有类名 dog 和 flower（因为图片中有一只狗和一些鲜花），而下面中间的图片拥有类名 motorcycle 和 water（确切地讲，是一辆雅马哈 V-Star 摩托车和科罗拉多河）。

在单击 Apply 之前，尝试将这些元素拖放到 Drop Zone 上。除了可以拖动图片外，没有发生什么事情。仔细观察事件指示器，注意它们也没有改变。这是意料之中的事情，因为一开始页面上并不存在可放置元素。

现在，请保持所有的控件处于初始状态（包括选中所有的 accept 复选框），单击 Apply 按钮。执行的命令包含一个 accept 选项，用来指定匹配所有 4 个类名的选择器。

再次尝试拖动任何一张图片到 Drop Zone，同时注意观察事件指示器。这一次，当你开始移动任何一张图片时，将看到 Activate 指示器会短暂地高亮显示（或者说是脉动），表明可放置元素已经注意到拖动操作已经开始（操作发生在可放置元素所接受的可拖动元素上），并且触发了 dropactivate 事件。

拖动图片来回进出 Drop Zone。相应的 dropover 和 dropout 事件会在合适的时间点被触发（通过相应的指示器显示）。现在，在 Drop Zone 的外部区域放下图片，观察 Deactivate 指示器的脉动效果。

最后，重复拖动操作，但是这一次当图片在 Drop Zone 上方时放下图片。Drop 指示器脉动（表明触发了 drop 事件）。另外请注意，Drop Zone 会显示最近放在它上面的图片。

- **练习 2**——取消选中所有的 accept 复选框，单击 Apply。这会导致 accept 选项包含一个空字符串，不会匹配任何元素。无论选择哪张图片，都没有回调事件指示器脉动，同时当你把一张图片在 Drop Zone 上放下时也不会有任何事情发生。没有一个有效的 accept 选项，Drop Zone 成了一块没用的砖头。（注意，这和省略 accept 选项是不同的，那样会使得所有的元素成为可接受的元素。）
- **练习 3**——只选中一个复选框，例如 flower，注意只有图片上有花朵的图片（页面知道哪些图片上有花朵，因为有花朵的图片定义有 flower 类名）才被认为是可接受的元素。

  再次尝试任何你所喜欢的可接受类名的组合，熟悉 accept 选项的概念。
- **练习 4**——重置所有控件，选中 activeClass 的 greenBorder 单选按钮，然后单击 Apply。这会为可放置元素提供 activeClass 选项，用来指定一个定义绿色边框的类名（你应该可以猜到）。

  现在，当你开始拖动一个被可放置元素所接受的图片（由 accept 选项定义）时，Drop Zone 周围的黑色边框会变为绿色边框。
- **练习 5**——重置实验室页面，选中标签为 bronze 的 hoverClass 单选按钮，然后单击 Apply。当拖动可接受图片到 Drop Zone 上时，Drop Zone 改变为古铜色。
- **练习 6**——在这个练习中，分别选中 tolerance 单选按钮中的每一个，观察这个设置是如

何影响可放置元素从活动到悬停的转变的（换句话说，就是什么时间触发 dropover 事件的）。这个转变可以通过设置 hoverClass 选项或者通过 Over 事件指示器的脉动来观察。

提示　如果你在自己网页上无法实现这个功能，需要注意 CSS 的优先级规则。当提供 activeClass 类名时，它必须能够覆盖赋予默认视觉效果（你希望替代的）的 CSS 规则。对于 hoverClass 也是同样的道理。（有时需要使用!important CSS 限定符来覆盖其他的样式规则。）

继续在实验室页面练习，确保你完全了解拖动和放置操作是如何运行的，以及如何被支持的选项所影响。

一旦可以进行拖放操作后，我们就可以设计出整个用户交互场景：使用拖放让用户直接操作页面元素这种实现方式即简单又直观。这些交互中，排序操作也是一种无处不在的应用，因此 jQuery UI 对此提供了直接的支持。

10.3　排序

可以说，排序是使用拖放操作最有用的交互之一。把列表中的项目按照特定顺序排列，甚至在列表之间移动项目，这在桌面应用中是一个相当普遍的交互技术。但是在 Web 上，要么缺少这种技术，要么使用<select>元素和按钮的组合来模拟这种技术（在列表中移动项目，有时在多个选择列表之间移动项目）。

尽管这种组合控件并不可怕，但是如果能让用户直接操作元素岂不是更加直观。拖放元素赋予了我们这种能力，而 jQuery UI 使得它更加容易使用。

与可拖动和可放置功能一样，jQuery UI 还通过一个简单的多用途 sortable()方法来提供排序操作，这个方法的语法现在看起来应该很熟悉了。

命令语法：sortable

```
sortable(options)
sortable('disable')
sortable('enable')
sortable('destroy')
sortable('option', optionName, value)
sortable('cancel')
sortable('refresh')
sortable('refreshPositions')
sortable('serialize')
sortable('toArray')
```

根据指定的选项使包装集中的元素可排序，或基于作为第一个参数传入的操作字符串来执行其他的可排序性操作

参数

options	（对象）一个散列对象，由要应用到包装集元素上的选项组成（参见表 10-5），从而使得这些元素可排序
'disable'	（字符串）禁用包装集中任何可排序元素的可排序性。并不删除元素的可排序性，可以通过调用这个方法的'enable'变体来恢复元素的可排序性
'enable'	（字符串）重新启用包装集中任何可排序元素（它们的可排序性已被禁用）的可排序性。注意这个方法不会向任何不可排序的元素添加可排序性
'destroy'	（字符串）删除包装集中元素的可排序性
'options'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是可排序元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
'cancel'	（字符串）取消当前的排序操作。这个方法在 sortreceive 和 sortstop 事件的处理器中非常有用
'refresh'	（字符串）触发可排序元素中的项目重新加载。调用这个方法将会导致添加到可排序列表中的新项目被识别出来
'refreshPositions'	（字符串）这个方法主要用于在内部刷新可排序元素的缓存信息。滥用会削弱排序操作的性能，因此只有在必要的时候才使用这个方法（用来解决过期缓存信息产生的问题）
'serialize'	（字符串）从可排序元素返回一个序列化的查询字符串（可以通过 Ajax 来提交）。我们很快就会更详细地讨论这个方法
'toArray'	（字符串）返回由可排序元素的 id 值组成的数组（按照元素的排列顺序）
optionName	（字符串）要设置或返回的选项名称的值（参见表 10-5）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
value	（对象）要设置的选项的值（通过 optionName 参数标识）

返回值

包装集，除了返回选项值的情况

与前面讲过的交互部件相比，sortable()方法的变体增加了不少，我们会详细考察其中的一些变体，不过首先来看看如何使得元素可排序。

10.3.1　使元素可排序

我们几乎可以使任何一组子元素成为可排序元素（通过向它们的父元素应用可排序性），但是我们通常将它应用到列表上（`<ul>`或`<ol>`元素），以便它的`<li>`子元素可以来回移动。这样不仅能增强语义，而且可以优雅地将可排序元素降级为普通的列表元素。

另一方面，如果应用程序需要，我们可以随意向父`<div>`元素的子`<div>`元素应用可排序性。我们在表 10-5 中考察可排序性选项时，将会看到如何做到这一点。

如可拖动和可放置交互元素一样，可排序性也是通过不带参数（接受默认值），或者带一个提供非默认值选项的对象参数来调用 `sortable` 方法。

jQuery UI 可排序实验室页面可以从 chapter10/sortables/lab.sortables.html 找到，我们可以使用这个页面来实时地观察可排序性选项的操作。这个实验室页面如图 10-4 所示。

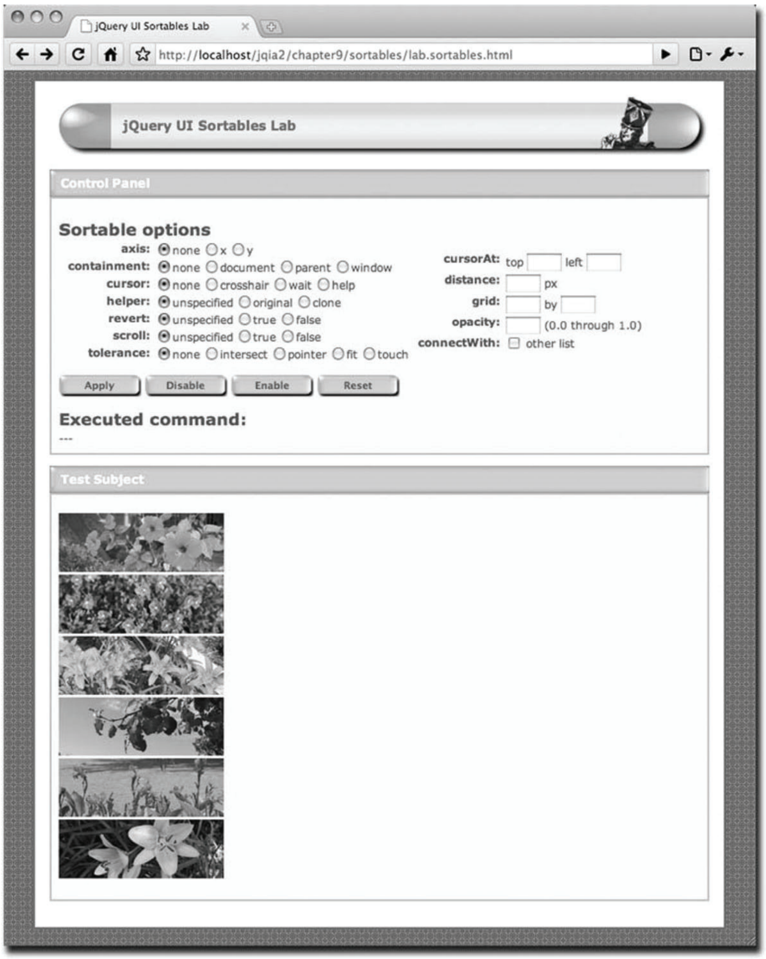


图 10-4　可排序实验室可以向列表应用各种可排序性选项

这没什么奇怪的，很多传入可排序元素的选项只是传入了更低层级的拖动或者放置操作。为了节约篇幅，就不再重复描述这些选项了，我们提供了描述这些选项的表格作为参考。

这些选项中的大部分都是不言自明的，但是需要重点关注 connectWith 选项。

表10-5 jQuery UI的**sortable()**方法的选项

选 项	描 述	是否在实验室页面中
activate	（函数）指定在可排序元素上创建的函数，用来作为sortactivate事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
appendTo	请参见表10-1中具有相同名称的可拖动操作的选项。追加为body元素的子元素可以解决重叠和z-index的相关问题	
axis	请参见表10-1中具有相同名称的可拖动操作的选项。通常，这个选项用来限定可排序列表的移动方向（水平或垂直）	✓
beforeStop	（函数）指定在可排序元素上创建的函数，用来作为sortbeforeStop事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
cancel	请参见表10-1中具有相同名称的可拖动操作的选项	
change	（函数）指定在可排序元素上创建的函数，用来作为sortchange事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
connectWith	（选择器）标识另一个可排序元素，该元素可以接受当前这个可排序元素中的项目。这将允许把元素从一个列表移动到另一个列表——经常会用到的用户交互。如果省略此选项，则不连接其他元素	✓
containment	请参见表10-1中具有相同名称的可拖动操作的选项	✓
cursor	请参见表10-1中具有相同名称的可拖动操作的选项	✓
cursorAt	请参见表10-1中具有相同名称的可拖动操作的选项	✓
deactivate	（函数）指定在可排序元素上创建的函数，用来作为sortdeactivate事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
delay	请参见表10-1中具有相同名称的可拖动操作的选项	
distance	请参见表10-1中具有相同名称的可拖动操作的选项	✓
dropOnEmpty	（布尔）如果为true（默认值），则允许从这个可排序列表拖放项目到连接的另一个没有项目的可排序列表。否则，不允许向没有项目的排序列表放置项目	
forceHelperSize	（布尔）如果为true，则强制助手具有一定尺寸。默认为false	
forcePlaceholderSize	（布尔）如果为true，则强制占位符具有一定尺寸。默认为false	
grid	请参见表10-1中具有相同名称的可拖动操作的选项	✓
handle	请参见表10-1中具有相同名称的可拖动操作的选项	
helper	请参见表10-1中具有相同名称的可拖动操作的选项	
items	（选择器）在可排序元素的上下文中，提供识别哪些子项目可以排序的选择器。默认使用选择器>*，它允许选择所有的子项目来排序	
opacity	请参见表10-1中具有相同名称的可拖动操作的选项	✓
out	（函数）指定在可排序元素上创建的函数，用来作为sortout事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
over	（函数）指定在可排序元素上创建的函数，用来作为sortover事件的事件处理器。有关这个事件的更多细节，请参见表10-6	

（续）

选　　项	描　　述	是否在实验室页面中
placeholder	（字符串）类名，应用到可能未定义样式的占位符空间上	
receive	（函数）指定在可排序元素上创建的函数，用来作为sortreceive事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
remove	（函数）指定在可排序元素上创建的函数，用来作为sortremove事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
revert	请参见表10-1中具有相同名称的可拖动操作的选项。当启用这个特效时，拖动助手将平滑地移入新位置，而不是快速地吸附到新位置	✓
scroll	请参见表10-1中具有相同名称的可拖动操作的选项	✓
scrollSensitivity	请参见表10-1中具有相同名称的可拖动操作的选项	
scrollSpeed	请参见表10-1中具有相同名称的可拖动操作的选项	
sort	（函数）指定在可排序元素上创建的函数，用来作为sort事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
start	（函数）指定在可排序元素上创建的函数，用来作为sortstart事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
stop	（函数）指定在可排序元素上创建的函数，用来作为sortstop事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
tolerance	请参见表10-3中具有相同名称的可放置操作的选项	✓
update	（函数）指定在可排序元素上创建的函数，用来作为sortupdate事件的事件处理器。有关这个事件的更多细节，请参见表10-6	
zIndex	请参见表10-1中具有相同名称的可拖动操作的选项	

10.3.2　连接可排序元素

允许用户对单个列表中的元素进行排序非常有用，这是显而易见的。不过，从一个列表向另一个列表移动项目也是一个常见的操作。这个功能通常是由两个多选列表和一个按钮的组合实现（把选中的项目从一个列表移动到另一个列表），为了控制每个列表中项目的顺序或许需要更多的按钮。

通过使用 jQuery UI 的可排序元素，并用 connectWith 选项将它们连接起来，就可以消除这些按钮，从而呈现给用户一个整洁的、可以直接操作的界面。试想一下，也许有一个页面，用户可以设计他们喜欢的报告格式。报告可能包含很多数据列，但是我们允许用户指定他们想要包含的列，以及这些列显示的顺序。

可以在列表中包含所有可能的列，并允许用户从列表中拖动他们需要的列到第二个列表中，第二个列表的内容按照报告列出现的顺序来呈现。

用来创建这个复杂交互的代码却非常简单，如下所示：

```
$('#allPossibleColumns').sortable({
  connectWith: '#includedColumns'
});
$('#includedColumns').sortable();
```

在可排序实验室页面，你可以选中标签为 connectWith 的复选框，试着在两个列表之间拖动元素。

在所有这些拖动和放置操作中（更不用说在列表内或列表之间拖动元素），我们想要深入了解的那些事件，以及在排序操作中到底发生了什么。

10.3.3 可排序事件

在排序操作中有很多移动和晃动，会触发拖放事件以及操作 DOM 元素——不仅为了来回移动可排序的元素，也为了处理可能已经定义的占位符。

如果我们所关心的只是允许用户对列表中的项目进行排序，然后再获取结果（将在下一节讨论），那么就不需要要特别关心在排序过程中发生的所有事件。但是，与可拖动和可放置元素一样，如果想要在感兴趣的事件发生时执行某些操作，那么可以定义处理器函数，以便在这些事件发生时收到提醒。

在其他交互元素中我们曾看到过,可以通过传入 sortable()的选项来当场创建这些处理器，也可以使用 bind()来创建这些处理器。

传入这些事件处理器的信息遵循通用的交互格式，将事件作为第一个参数，将自定义对象（其中包含感兴趣的操作信息）作为第二个参数。对于可排序元素，这个自定义对象包含如下属性。

- position——对象，其 top 和 left 属性给出了被拖动元素相对于偏移父元素的位置。
- offset——对象，其 top 和 left 属性给出了被拖动元素相对于文档页面的位置。
- helper——包含拖动助手元素的包装集（通常是一个副本）。
- item——包含排序项目的包装集。
- placeholder——包含占位符（如果存在的话）的包装集。
- sender——当连接操作在两个可排序元素之间发生时，用来指示包含发起连接操作的可排序元素的包装集。

注意，如果这些属性在一些状态下没有意义的话，它们的值可能是 undefined 或 null。例如，sortstop 事件没有对 helper 的定义，因为此时拖动操作已经停止了。

这些处理器的函数上下文是应用了 sortable()方法的元素。

在排序操作中触发的事件列表如表 10-6 所示。

表10-6 为可排序部件触发的jQuery UI事件

事 件	选 项	描 述
sort	sort	在排序操作中为mousemove事件重复触发
sortactivate	activate	当排序操作发生在发起连接的可排序元素上时触发
sortbeforestop	beforeStop	在排序操作将要结束时触发，此时助手和占位符元素的引用仍然可用
sortchange	change	在排序元素在DOM中改变位置时触发

（续）

事　　件	选　　项	描　　述
sortdeactivate	deactivate	当连接的排序停止时触发，并将此事件传播到相连接的可排序列表
sortout	out	当排序项目移出连接的列表时触发
sortover	over	当排序项目移入到连接的列表时触发
sortreceive	receive	当连接的列表从其他列表接收到排序项目时触发
sortremove	remove	当排序项目从连接的列表中删除并被拖放到另一个列表时触发
sortstart	start	当排序操作开始时触发
sortstop	stop	当排序操作结束后触发
sortupdate	update	当排序操作停止并且列表中项目的位置发生改变时触发

注意，这些事件类型中的很多只在涉及连接的列表操作时才会触发。只在单个列表排序操作中触发的事件比较少。

sortupdate 事件可能是最重要的，因为当排序真的发生改变时它会通知我们。如果一个排序操作发生了但是没有任何改变，则这种情况可能不需要关心。

当 sortupdate 事件发生时，我们可能想知道排序后的列表顺序。下面就来介绍如何得到这个信息。

10.3.4　获取排序的顺序

当我们想知道可排序列表中的顺序时，可以使用 sortable()方法的两个变体，使用哪个取决于需要什么类型的数据。

sortable('toArray')方法返回包含排序项目 id 值的 JavaScript 数组（按照项目的排列顺序）。如果需要知道项目的顺序，可以使用这个方法。

另外，如果想要将此信息作为新请求的一部分来提交，甚至放在一个 Ajax 请求中，那么可以使用 sortable('serialize')，它会返回一个适合作为查询字符串或请求主体的字符串，其中包含排序元素的排列信息。

使用这个变体需要为可排序元素（这个元素是将要被排序的元素，而不是可排序元素本身）的 id 值使用一种特别的格式。每个 id 需要采用 prefix_number（前缀_数字）的格式，其中前缀可以是任意字符串（只要所有被排序的元素一样就行），紧跟着一个下划线和一个数字值。当遵守这种格式时，序列化可排序元素的结果会生成一个查询字符串，其中为每个可排序元素包含一个项目，项目的名称是前缀紧跟着[]，项目的值是 id 值尾部的数字值。

困惑了？不要责怪自己。下面转到可排序实验室页面来寻求帮助。

实验室页面底部的控制面板（在图 10-4 中看不到，因为它超出了屏幕显示范围）显示了在排序操作结束后（使用 sortupdate 事件处理器）调用获取方法的结果。分配给排序元素的 id 值从上到下依次是 subject_1 到 subject_6，遵循 serialize 方法定义的格式规则。

在实验室中，使所有选项处于默认状态，单击 Apply，点住橙色的百合花图片（`id` 值为 `subject_3`），并拖动它使其成为列表中的第一项。在控制台面板，你将观察到 `id` 值形成的数组如下所示：

```
['subject_3', 'subject_1', 'subject_2', 'subject_4', 'subject_5',
     'subject_6']
```

这正是我们所期望的，显示了项目的新顺序（其中第三个项目现在位于第一个位置）。

序列化的结果如下：

```
subject[]=3&subject[]=1&subject[]=2&subject[]=4&subject[]=5&subject[]=6
```

这说明前缀（`subject`）是用来构成查询参数的名称，而尾部的数字值将成为参数值。（顺便提一下，`[]`是表示“数组”的常用记号，它用来表明有多个同名参数。）

如果这种格式不是你偏好的类型，那么可以使用 `id` 值组成的数组为基础来创建自己的查询字符串（在这种情况下`$.param()`会相当的方便）。

作为练习，请回到第 5 章有关“可折叠模块”的示例（当时我们只关注使用动画的方式将主体内容卷入到标题栏中）。该如何使用可排序元素让用户在多个列中管理这些模块（有些人也称之为门户页面）的位置?

在可排序元素中，基本的拖动和放置交互已经被整合在一起，以便提供更高级别的交互元素。下面就来看看 jQuery UI 提供的另外一个类似的交互元素。

10.4　改变元素的尺寸

在第 3 章，我们学习了如何使用 jQuery 的方法来改变 DOM 元素的尺寸。在第 5 章，我们介绍了如何以动画的方式来完成这个功能。jQuery UI 也允许用户直接改变元素的尺寸。

再次思考可折叠模块示例，除了允许用户在页面上移动那些模块，如果能够让用户轻松地自定义模块的尺寸岂不是更好?

目前为止看到的所有交互元素，都不需要在页面上包含 jQuery UI 下载包中生成的 CSS 文件。但是为了让可改变尺寸交互元素正常运行，就必须向页面导入 CSS 文件，代码如下所示：

```
<link rel="stylesheet" type="text/css"
      href="styles/ui-theme/jquery-ui-1.8.custom.css">
```

无需详细说明，使用 `resizable()`方法和使用其他 jQuery UI 交互元素一样简单，它的语法模式大家应该很熟悉了。

这里也没有什么新东西（这些交互方法模式你应该相当熟悉了），因此下面就来看看创建可改变尺寸元素时可用的选项。

命令语法：resizable

resizable(options)
resizable('disable')
resizable('enable')
resizable('destroy')
resizable('option', optionName, value)

根据指定的选项使包装集中的元素可改变尺寸，或者基于作为第一个参数传入的操作字符串来执行其他的可改变尺寸的操作

参数

`options`	（对象）一个散列对象，由要应用到包装集元素上的选项组成（参见表 10-7），从而使得这些元素可改变尺寸
`'disable'`	（字符串）暂时地禁用包装集中任何可改变尺寸元素的可改变尺寸性。并不删除元素的可改变尺寸性，可以通过调用这个方法的`'enable'`变体来恢复元素的可改变尺寸性
`'enable'`	（字符串）重新启用包装集中任何可改变尺寸元素（它们的可改变尺寸性已被禁用）的可改变尺寸性。注意，这个方法不会向任何不可改变尺寸的元素添加可改变尺寸性
`'destroy'`	（字符串）删除包装集中元素的可改变尺寸性
`'option'`	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是可改变尺寸的元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 `optionName` 参数
`optionName`	（字符串）要设置或返回的选项名称的值（参见表10-7）。如果提供 `value` 参数，则这个值就成为选项的值。如果没有提供 `value` 参数，则返回已命名选项的值
`value`	（对象）要设置的选项的值（通过 `optionName` 参数来标识）

返回值

包装集，除了返回选项值的情况

10.4.1　使元素可改变尺寸

一种选择很少能适合所有的情况，因此，和其他的交互方法一样，`resizable()`方法也提供了很多选项，可用来自定义各种所需的交互操作。

jQuery UI 可改变尺寸实验室页面的地址是 chapter10/resizables/lab.resizables.html，如图 10-5 所示。

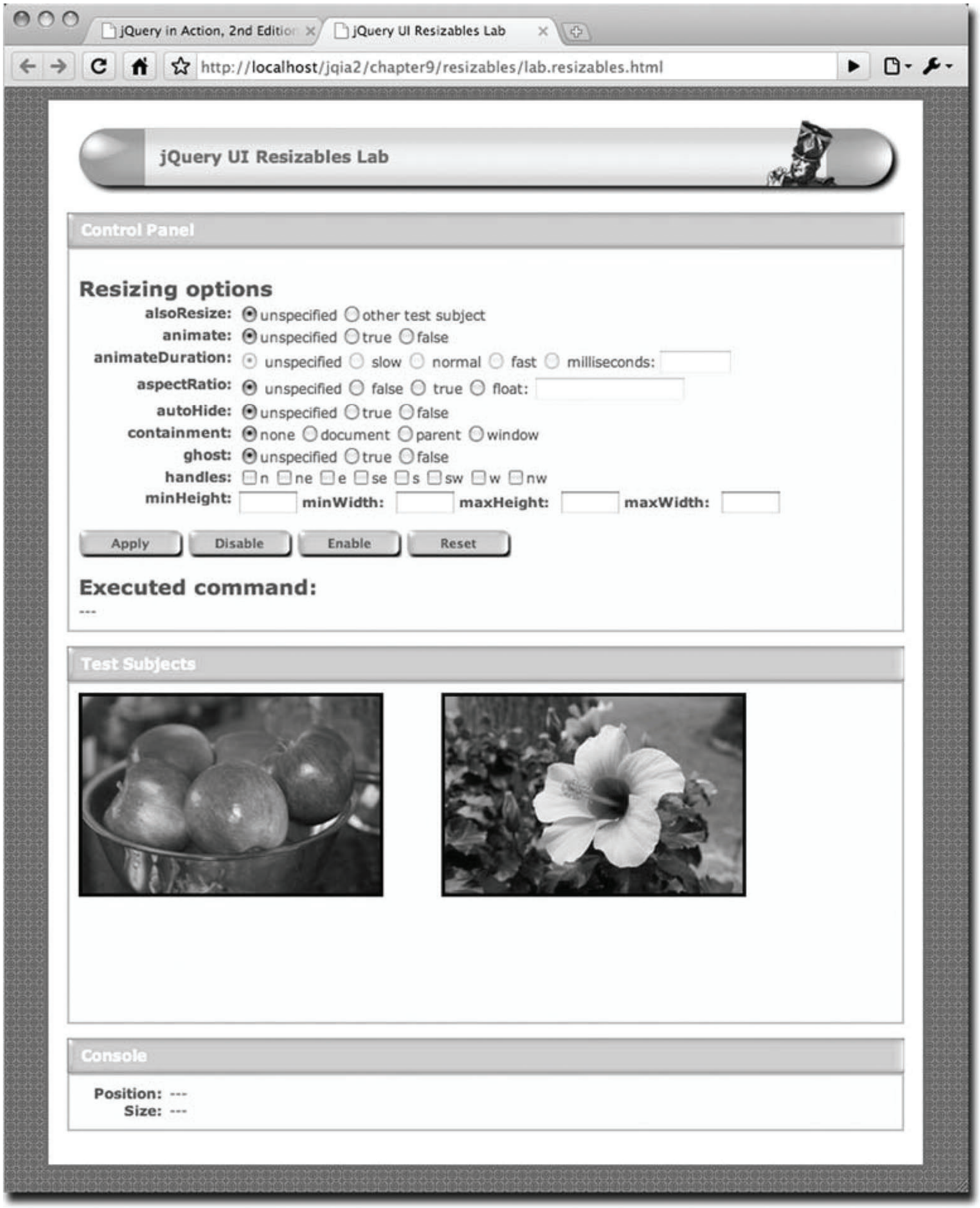


图 10-5　可改变尺寸实验室可以让我们实时观察各种可改变尺寸选项的操作

这个实验室的 Control Panel 可以尝试 `resizable()`方法大部分选项。当你在阅读表 10-7 中的选项列表时可以在这个实验室中进行操作。

与一些更加复杂的交互元素相比，resizable()元素的选项集很小。其实它的事件也很少，后面将会介绍。

表10-7　jQuery UI的**resizable()**方法的选项

选　　项	描　　述	是否在实验室页面中
alsoResize	（选择器|jQuery|元素）指定与可改变尺寸的元素同步改变尺寸的其他DOM元素。不需要向这些DOM元素应用resizable()方法。如果省略，则没有任何其他的元素会受到影响	✓

animate	（布尔）如果为true，在拖动操作结束之前，元素本身不会改变尺寸，操作结束后，元素会使用动画的方式平滑地改变尺寸。在拖动过程中，将使用带有类ui-resizable-helper（除非通过helper选项来覆盖，下面会讨论）的助手显示拖动的轮廓。确保这个类拥有合适的CSS定义，否则在以动画方式改变尺寸的操作中你可能看不到任何效果
	例如，可改变尺寸实验室页面使用如下定义：
	`.ui-resizable-helper {`
	`  border: 1px solid #82bf5a;`
	`}`
	默认情况下，操作不带动画效果
		✓

animateDuration	（整数|字符串）如果启用animate选项，定义动画的持续时间。可以使用标准的动画字符串slow、normal或fast，或者以毫秒数指定的数字值	✓
animateEasing	（字符串）如果启用animate选项，指定使用的缓动特效。默认值是内置的swing缓动特效。请参阅第5章中有关缓动特效的讨论	
aspectRatio	（布尔|浮点数）指定是否或以什么比例来在改变尺寸操作中保持元素的长宽比。值为true会强制使用元素的原始长宽比，而浮点值可以用来指定公式width/height计算的比例。例如，3/4的比例可以指定为0.75 默认情况下，在操作中不会保持元素的长宽比	✓
autoHide	（布尔）如果为true，则手柄是隐藏的，只有当鼠标悬停在可改变尺寸元素上方时才显示手柄。参见handles选项以了解更多信息。默认情况下，总是显示手柄	✓
cancel	（选择器）指定排除在可改变尺寸操作之外的元素。默认情况下，使用选择器":input,option"	
containment	（字符串|元素|选择器）定义一个限制改变尺寸操作的区域。可以指定为内置的字符串parent或document，可以提供一个特别的元素，也可以使用选择器来获取容器元素。默认情况下，不限制改变尺寸操作	✓
delay	请参见表10-1中具有相同名称的可拖动操作的选项	
distance	请参见表10-1中具有相同名称的可拖动操作的选项	
ghost	（布尔）如果为true，在改变尺寸操作中显示一个半透明的助手元素。默认值为false	✓
grid	请参见表10-1中具有相同名称的可拖动操作的选项	

（续）

选　项	描　述	是否在实验室页面中
handles	（字符串|对象）指定元素可以在哪些方向上改变尺寸。可以指定为用逗号分隔的列表字符串，可能的值有：n、ne、e、se、s、sw、w、nw或all。如果想要jQuery UI来处理创建手柄的过程，可以使用这种格式 如果你想要使用可改变尺寸元素的子元素作为手柄的话，可以提供一个对象，其属性为能为八个方向分别定义手柄：n、ne、e、se、s、sw、w和nw。属性的值应该是选择器（用来定义作为手柄的元素） 我们将在讨论事件后更进一步了解这个手柄选项 如果省略，则为e、se和s三个方向创建手柄	✓
helper	（字符串）如果指定此选项，则在改变尺寸的操作中启用带有提供的类名的助手元素。可以使用这个选项来启用助手，也可以通过其他选项例如ghost或animate来隐式地启用助手。如果隐式地启用助手，则默认类名为ui-resizable-helper，除非使用这个选项来覆盖类名	
maxHeight	（整数）指定元素的尺寸可以调整到的最大高度。默认情况下，不设置最大值	✓
maxWidth	（整数）指定元素的尺寸可以调整到的最大宽度。默认情况下，不设置最大值	✓
minHeight	（整数）指定元素的尺寸可以调整到的最小高度。默认情况下是10个像素	✓
minWidth	（整数）指定元素的尺寸可以调整到的最小宽度。默认情况下是10个像素	✓
resize	（函数）指定在可改变尺寸元素上创建的函数，用来作为resize事件的事件处理器。有关这个事件的更多细节，请参见表10-8	✓
start	（函数）指定在可改变尺寸元素上创建的函数，用来作为resizestart事件的事件处理器。有关这个事件的更多细节，请参见表10-8	✓
stop	（函数）指定在可改变尺寸元素上创建的函数，用来作为resizestop事件的事件处理器。有关这个事件的更多细节，请参见表10-8	✓

10.4.2　可改变尺寸事件

只有 3 个简单的事件会在改变尺寸操作中触发，分别表示改变尺寸操作已经开始、进行中，以及已经结束。

传入这些处理器的信息遵循通用的交互格式，将事件作为第一个参数，将一个自定义对象（其中包含我们感兴趣的操作信息）作为第二个参数。对于可改变尺寸元素，这个自定义对象包含如下属性。

- position——一个对象，其 top 和 left 属性给出了元素相对于它偏移父元素的位置。
- size——一个对象，其 width 和 height 属性给出了元素的当前尺寸。
- originalPosition——一个对象，其 top 和 left 属性给出了元素相对于它偏移父元素的原始位置。
- originalSize——一个对象，其 width 和 height 属性给出了元素的原始尺寸。
- helper——包含所有助手元素的包装集。

10

注意，如果这些属性在一些状态下没有意义的话，则它们的值可能是 `undefined` 或 `null`。例如，可能没有定义 `helper`[1]。

这些处理器的函数上下文是应用 `resizable()`方法的元素。在改变尺寸操作中触发的事件如表 10-8 所示。

表10-8 为可改变尺寸元素触发的jQuery UI事件

事 件	选 项	描 述
resizestart	start	操作开始时触发
resize	resize	在操作中为mousemove事件重复触发
resizestop	stop	当操作结束时触发

可改变尺寸实验室使用这些事件来在页面的 Console 面板中报告测试对象元素的当前位置和尺寸。

10.4.3 为手柄添加样式

虽然 `resizable()`是一个相当简单的操作，但是至少就 jQuery UI 交互部件而言，还需要对手柄进行一些特别的讨论。

默认情况下，可改变尺寸部件会为东、东南以及南三个方向创建手柄，并启用这些方向的可改变尺寸功能。没有定义改变尺寸手柄的方向不能进行改变尺寸操作。

最初你可能会迷惑，在页面和可改变尺寸实验室页面中，无论启用多少个方向，只有东南角会出现一个“把手”图标。然而，当启用除东南角外的所有其他指定的方向时，它们工作得很好，并且当鼠标悬停在可改变尺寸的边界时会改变光标的形状。为什么会出现这种差异呢？

这不是你所能控制的代码。jQuery UI 把那个角落视为一种特殊情况，除了将东南方向的手柄和其他方向的手柄做同样处理外，还向其添加了额外的类名。

当创建手柄时，都会向其添加类名 `ui-resizable-handle` 和 `ui-resizable-`*`xx`*，其中 *xx* 代表手柄所呈现的方向（例如，`ui-resizable-n` 表示东边的手柄）。东南角被 jQuery UI 视为特殊情况，也接受类名 `ui-icon` 和 `ui-icon-gripsmall-diagonal-se`（在下载 jQuery UI 过程中由默认的 CSS 文件生成），用来创建显示在角落的“手柄”图标。尽管你可以操作手柄名称的 CSS 来影响所有的手柄（包括东南角的手柄）的显示方式，但是没有选项可以用来改

① 原文错误，传入 `resizable` 事件处理器第二参数的 `helper` 属性在任何情况下都不可能为 `null` 或 `undefined`，可以在实验室页面通过执行如下代码来验证：

```
$('.testSubject').resizable({
    start: function(event, ui) { console.log(ui.helper); },
    resize: function(event, ui) { console.log(ui.helper); },
    stop: function(event, ui) { console.log(ui.helper); }
});
```

变类名的分配行为[①]。

注意 毫无疑问，那个特殊的东南角“手柄”图标的灵感来自源于类似 Mac OS X 的窗体管理器，它们会在可改变尺寸的窗体上放置这样一个手柄。

图 10-6 显示了这个“把手”手柄，以及当鼠标悬停在可改变尺寸的东部边界附近时显示的 CSS 手柄。

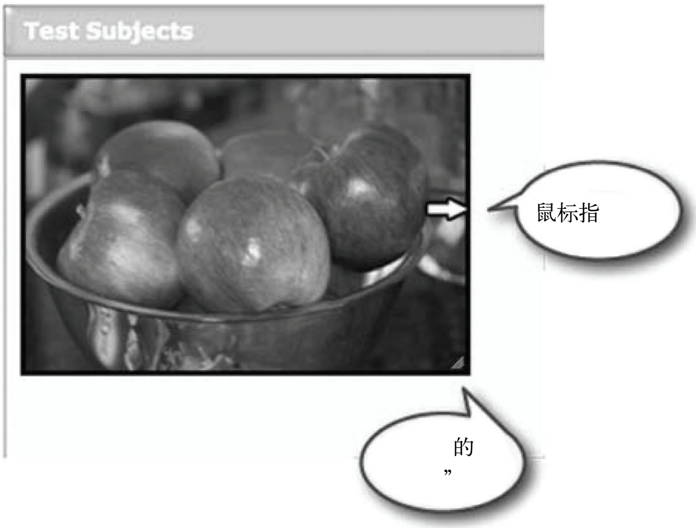


图 10-6 默认情况下，jQuery UI 在东南角放置一个“把手”手柄，并为其他手柄使用 CSS 指针

如果你觉得这种方式的限制太多的话，那么可以使用 handles 选项更加复杂的版本来定义作为手柄的子元素（可以自己创建这些元素）。

接下来看看 jQuery UI 提供的最后一个交互部件。

10.5 使元素可选择

目前所研究的大部分交互元素都涉及对元素的直接操作，目的是以某种方式改变元素的状态，从而影响它们在 DOM 中的位置、大小和顺序。selectable()交互元素能够设置和清空任何 DOM 元素的“选中”状态。

① 指的是可改变尺寸部件会自动向东南角的元素添加 ui-icon 和 ui-icon-gripsmall-diagonal-se 两个类名，这是无法禁用的默认行为。

在 HTML 表单控件中，我们经常使用类似复选框、单选按钮和<select>元素来保持选中的状态。jQuery UI 允许我们在元素上保持这种状态而不是借助于这些控件。

回想一下第 4 章介绍的 DVD Ambassador 示例。在该示例中，我们专注于筛选控件集，而没有特别留意从筛选操作返回的结果。我们将做出改变。作为提示，这个示例的截图如图 10-7 所示。



图 10-7　再次访问 DVD Ambassador 示例，并使用 jQuery UI 可选择部件来设置结果数据集

结果以表格的形式显示在页面上的元素列表中（在这个示例中，结果是一个硬编码的 HTML 片段，在真实的应用中结果将会从数据库信息中生成）。

比方说，我们想允许 DVD Ambassador 的用户选择一个或多个 DVD 标题并向其应用某些批量操作：例如从数据库删除它们，或者将它们标记为已读或未读。

传统的方式可能是为每一行添加一个复选框控件，并用它来指示是否选中该行。这种方法经证明是可以工作的，但是因为复选框的区域非常小，通常是 12 像素 × 12 像素，用户必须单击这个区域来切换选中状态，那些笨拙的单击者一定会叫苦不迭。

注意　在表单中，<label>元素经常用来将文本和复选框关联起来，从而使复选框和标签都可单击。

用户界面不应该成为训练手和眼睛协调性的游戏，我们想让用户更加容易地操作页面。我们拥有为整行设置单击处理器的诀窍，这样用户单击行的任意地方时，处理器都可以找出内部的复选框并切换它的值。这为用户提供了更大的单击目标，而复选框仅仅是作为视觉提示以及记住选中状态的手段。

jQuery UI 的可选择部件将这个功能发扬光大，它有两个特别的优点：

- 可以剔除复选框；
- 用户可以通过单个交互进行批量选择。

剔除复选框意味着需要对可选择的行（没有了复选框，用户就不知道哪些行可以选择，我们必须弥补这个缺陷）和行的选中状态提供自己的视觉提示。当单击某一行时，通过改变行的背景色来指示状态的变化是一种约定俗成的手段，同时通过改变光标来指示某些有趣的事情将要发生也不失为一个好主意。

至于记住选中了哪些行，没有选中哪些行，jQuery UI 可选择部件会为选中的元素添加一个类名（名为 `ui-selected`）来维持选中的状态。

使用复选框的方法，用户只能通过逐行单点复选框来选择或反选元素。尽管通常会提供一个复选框来切换所有复选框的状态，但是如果用户想选择第 3 行到第 7 行，该怎么办？他们只能被迫一行一行地单击复选框。

使用可选择部件，我们不仅可以通过单击来单选，而且可以通过跨元素拖出一个矩形选取框（或者拖出一个包含要选择元素的矩形区域，取决于如何设置选项[①]）来一次选择多个相邻的元素——更像是我们经常在很多桌面应用中操作的那样。

另外，可选择部件允许在单击或者拖动过程中按下 Ctrl 键（Mac 下是 Command 键）不放，从而向已经选择的元素集合中*添加*新元素。

现在可以打开 jQuery UI 可选择实验室页面，该页面如图 10-8 所示，文件地址是 chapter10/selectables/lab.selectables.html。在这个实验室中，我们使用 DVD Ambassador 表格形式的结果集作为测试对象。

下面使用默认的选项来尝试一些交互操作。

- **练习 1**——在做任何改变和单击任何按钮之前，先在数据表格上移动鼠标光标，并尝试单击或者在多个行上拖动鼠标。注意光标的外观没有发生任何改变，而且在单击时没有产生任何效果。拖动只会选中浏览器中的文本。

  让选项保持它们的默认设置，单击 Apply 按钮。现在注意当鼠标光标悬停在任意数据行时它的外观改变成了手形指针。当 DOM 元素成为可选择的元素时（有资格被选择），jQuery UI 会在那个元素上放置类 `ui-selectee`。在实验室页面，下面的 CSS 规则导致了光标的变化：

```
.ui-selectee { cursor: pointer; }
```


① 这里指的是 `tolerance` 选项，后面会有详细介绍。

现在单击一些行，注意每当单击一行时，该行的背景色都会发生改变。当选中某个元素时，jQuery UI 会向其应用类 `ui-selected`，实验室页面使用下面的规则来改变选中元素的背景色：

```
#testSubject .ui-selected { background-color: pink; }
```

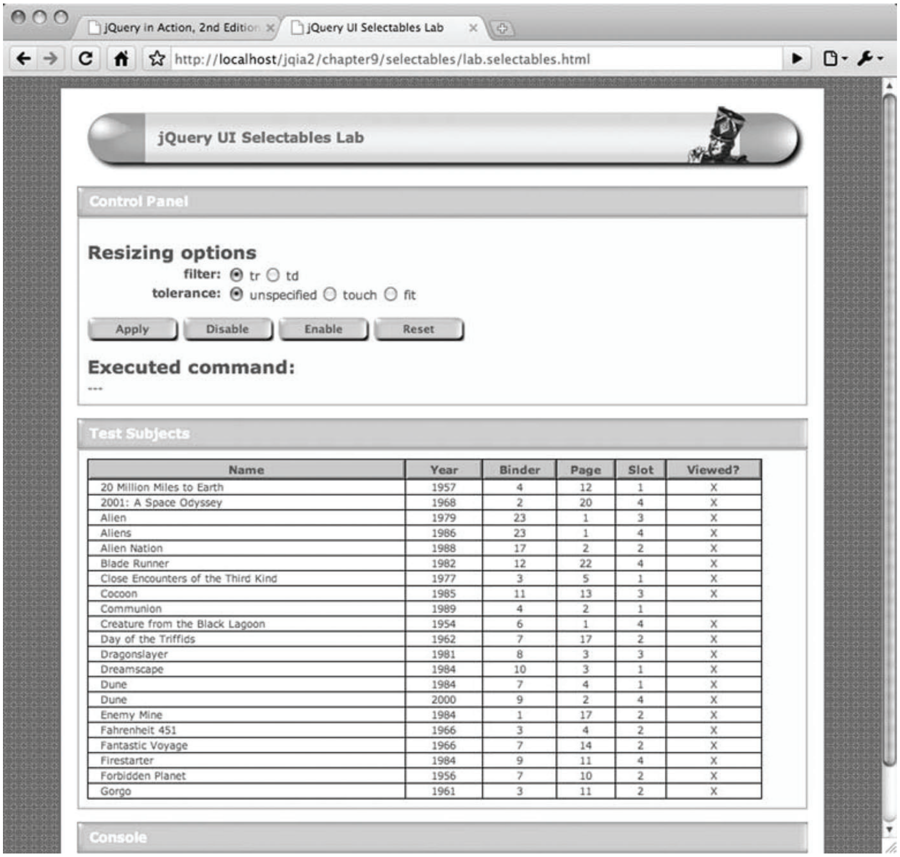


图 10-8　可选择实验室使用 DVD Ambassador 中的 HTML 结果片段作为测试对象

注意每当单击一行时，在这一行被选中的同时，任何之前选中的行会变为未选中状态。

- **练习 2**——不要改变任何设置或者单击任何按钮，选中一行，然后当选择其他行时按下 Ctrl/Command 键不放。注意，当单击时按下 Ctrl/Command 键会使之前选中的元素仍然处于选中状态。
- **练习 3**——不要改变任何设置或单击任何按钮，开始一个跨多行的矩形拖动操作。确保拖动操作从行的内部开始。注意拖动操作跨越的所有行都变成选中状态。同样地，在拖动操作中按住 Ctrl/Command 键不放会使之前选中的行保持选中状态。

目前这些都是可以完成的。

10.5.1 创建可选择元素

现在，我们已经通过示例看到了可选择部件的运行效果，下面就来看看示例所使用的selectable()方法。

命令语法：selectable

selectable(options)
selectable('disable')
selectable('enable')
selectable('destroy')
selectable('option', optionName, value)
selectable('refresh')

根据指定的选项使包装集中的元素可选择，或者基于作为第一个参数传入的操作字符串来执行其他的可选择操作

参数

options	（对象）一个散列对象，由要应用到包装集元素上的选项组成（参见表10-9），从而使得这些元素可选择
'disable'	（字符串）暂时地禁用包装集中任何可选择元素的可选择性。并不删除元素的可选择性，可以通过调用这个方法的'enable'变体来恢复元素的可选择性
'enable'	（字符串）重新启用包装集中任何可选择元素（它们的可选择性已被禁用）的可选择性。注意这个方法不会向任何不可选择的元素添加可选择性
'destroy'	（字符串）删除包装集中元素的可选择性
'refresh'	（字符串）更新可选择元素的尺寸和位置。主要用于禁用autoRefresh选项的情况
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是可选择的元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供optionName参数
optionName	（字符串）要设置或者返回的选项名称的值（参见表10-9）。如果提供value参数，则这个值就成为选项的值。如果没有提供value参数，则返回已命名选项的值
value	（对象）要设置的选项的值（通过optionName参数来标识）

返回值

包装集，除了返回选项值的情况

创建可选择元素可使用的选项参见表10-9。

表10-9　jQuery UI的**selectable()**方法的选项

选　　项	描　　述	是否在实验室页面中
`autoRefresh`	（布尔）如果为true（默认值），则在选择操作开始的时候计算每个可选择项目的位置和尺寸 尽管可选择操作不会对可选择元素的位置和尺寸做任何改变，但是它们可能会被页面上CSS或脚本改变 如果有很多可选择元素，出于性能考虑可以禁用这个选项，然后使用refresh方法来手工计算这些值	
`cancel`	请参见表10-1中具有相同名称的可拖动操作的选项	
`delay`	请参见表10-1中具有相同名称的可拖动操作的选项	
`distance`	请参见表10-1中具有相同名称的可拖动操作的选项	
`filter`	（选择器）指定选择器来找出包装集中哪些类型的子元素会成为可选择元素。所有这些元素都被添加了类ui-selectee 默认情况下，所有的子元素都可以作为可选择元素	✓
`selected`	（函数）指定在可选择元素上创建的函数，用来作为selectableselected事件的事件处理器。有关这个事件的更多细节，请参见表10-10	✓
`selecting`	（函数）指定在可选择元素上创建的函数，用来作为selectableselecting事件的事件处理器。有关这个事件的更多细节，请参见表10-10	✓
`start`	（函数）指定在可选择元素上创建的函数，用来作为selectablestart事件的事件处理器。向这个处理器传入事件实例，但是不会传递其他信息。有关这个事件的更多细节，请参见表10-10	✓
`stop`	（函数）指定在可选择元素上创建的函数，用来作为selectablestop事件的事件处理器。向这个处理器传入事件实例，但是不会传递其他信息。有关这个事件的更多细节，请参见表10-10	✓
`tolerance`	（字符串）fit或touch（默认值）其中之一 如果是fit，则拖动选择必须完全包围一个元素才能选择这个元素。在一些布局中可能会存在问题，因为拖动选择必须从一个可选择元素内部开始 如果是touch，则拖动矩形只需要和可选择项目的任何部分相交就能选中这个元素	✓
`unselected`	（函数）指定在可选择元素上创建的函数，用来作为selectableunselected事件的事件处理器。有关这个事件的更多细节，请参见表10-10	✓
`unselecting`	（函数）指定在可选择元素上创建的函数，用来作为selectableunselecting事件的事件处理器。有关这个事件的更多细节，请参见表10-10	✓

现在，我们已经介绍了这些选项，下面就来使用可选择实验室尝试些其他练习。

- **练习 4**——重复练习 1 到练习 3 的动作，这次请注意观察页面底部的 Console 面板。这个面板会显示选择操作中发生的事件。我们会在下一节讨论有哪些信息传入这些事件。
- **练习 5**——目前为止的所有练习中，我们指定 `filter` 选项为 `tr`，这使得整个数据行成为可选择元素。单击 Reset 按钮或刷新页面，为 `filter` 选项选择 `td` 值，单击 Apply。

在数据表格中单击来选择不同的元素。注意现在可以选择单个的数据单元格而不是整行数据。

- **练习 6**——改变 `filter` 选项的值为 `span`，单击 Apply。现在单击数据结果中各种不同的文本值。注意，这会只选择文本本身而不是整个单元格。（`<td>`元素中的每个文本值都包含在一个`<span>`元素中。）
- **练习 7**——重置页面，选择 `tolerance` 选项的值为 `touch`，单击 Apply。尝试各种不同的拖动操作，注意行为并没有发生改变，和选取框重叠的任何行都将被选择。
- 现在改变 `filter` 值为 `td` 并重复练习，注意任何与选取框相交的元素都会被选中。
- **练习 8**——保持 `filter` 的值为 `td`，选择 `tolerance` 的值为 `fit`，单击 Apply。重复拖动练习，注意只有完全被选取框包围的单元格才会被选中。

现在改变 `filter` 的值为 `tr`，单击 Apply，再次尝试。有什么新发现吗？

因为拖动选择必须从一个可选择元素的内部开始，`tolerance` 的设置[①]需要可选择元素完全被选取框包围才能被选中，而数据行没有被其他可选择元素所包围，因此这个组合将无法选取任何行。又上了一课？因此使用 `tolerance` 的 `fit` 值时一定要小心。

`selectable()`的选项列表比其他交互元素的选项列表要短。事实上，大部分都是为可选择事件创建事件处理器的快捷方式。不过这些事件是可选择操作过程中的重要组成部分。下面就来研究这些事件。

10.5.2 可选择事件

这么一个看似简单的操作，却在一个可选择操作中触发了一系列的事件集。不仅有用来识别操作开始和结束的事件，而且有选择或反选单个元素的事件，甚至有元素选择状态即将改变之前的事件。

和其他的交互元素不同，可选择事件没有传入处理器的固定结构。相反，如果有任何事件信息的话，都会根据不同的事件类型进行调整后再传入处理器。表 10-10 描述了可选择事件以及传入这些事件的数据。

如果你对某些事件不了解的话，特别是类似 `selectableselecting` 和 `selectableselected` 事件的区别，请重复可选择实验室的练习，仔细观察 Console 面板以了解在执行不同类型的选择操作时是如何触发事件的。

好了，我们已经选择了需要的元素。现在该怎么办呢？

表10-10 jQuery UI为可选择元素触发的事件

事　　件	选　　项	描　　述
`selectablestart`	`start`	可选择操作开始时触发。向事件处理器传递的第一个参数是事件实例，第二个参数是一个空对象

① 这里指的是将 `tolerance` 设置为 `fit` 的情况。

（续）

事 件	选 项	描 述
selectableselecting	selecting	为即将被选择的每个可选择元素触发。向事件处理器传递的第一个参数是事件实例，第二个参数是一个包含单个属性selecting的对象，它是一个对即将选择的元素的引用 向这些元素添加类名ui-selecting。如果存在类名ui-unselecting，就删除这个类名 这个事件所报告的元素最终并不一定会被选择。如果用户拖出一个包含某个元素的选取框，则这个元素就会被这个事件所报告。如果选取框发生改变导致不再包含这个元素，则这个元素就不会被选中
selectableselected	selected	为每个选中的元素触发。向事件处理器传递的第一个参数是事件实例，第二个参数是包含单个属性selected的对象，它是对已选择的元素的引用 向这些元素添加类名ui-selected并删除类ui-unselected
selectableunselecting	unselecting	为即将不被选择的每个可选择元素触发。向事件处理器传递的第一个参数是事件实例，第二个参数是包含单个属性unselecting的对象，它是对即将取消选择的元素的引用 向这些元素添加类名ui-unselecting 和selecting事件一样，这个事件所报告的并不总是会成为被选中的元素
selectableunselected	unselected	为每个不被选择的元素触发。向事件处理器传递的第一个参数是事件实例，第二个参数是包含单个属性unselected的对象，它是对不被选择的元素的引用 从这些元素上删除类名ui-unselecting
selectablestop	stop	可选择操作结束时触发。事件实例是传入这个事件处理器的唯一参数

10.5.3 查找已选择的和可选择的元素

可选择元素事件中最经常绑定的事件可能是selectablestop，它会在选择事件发生和结束后通知我们。在这个事件的处理器中，我们总是希望确定最终选择了哪些元素。

一般我们不会关心选择发生的用户选中项，但需要知道最终选中了哪些项，例如当需要和服务器通信的时候。

通常使用保持状态的 HTML 控件将状态作为表单提交的一部分，而无需我们提供任何帮助。但是如果需要将可选择元素的选择项作为表单提交的一部分来进行交流，甚至是作为 Ajax 请求的参数，就需要收集选择了哪些项。

你可能会回想起可排序交互部件提供了几个方法，可以确定可排序元素的最终状态。令人失望的是可选择部件不具备相同的功能。

但是失望也只是暂时的。可以使用 jQuery 选择器轻松地获取选择的元素，因此使用一个专门的方法来获取这些元素就没有必要了。因为每个选择的元素都被标记了类名 `ui-selected`，所以获取包含所有选择元素的包装集将会非常容易，如下所示：

```
$('.ui-selected')
```

如果只想获取选择的`<div>`元素，那么可以这么做：

```
$('div.ui-selected')
```

很多时候我们想做的可能是收集选择的元素，以便将选择项传输到服务器，这与复选框或者单选按钮使用请求参数来传输数据一样。如果想让选择项作为表单提交的一部分向服务器提交，一个简单的方法是在提交之前向表单添加隐藏的`<input>`元素（为每个选择的元素添加一个隐藏字段）。

在可选择实验室中，假设我们想要将所有选择的电影名称作为名为`title[]`的请求参数数组来提交。可以使用位于表单提交处理器中的如下代码来实现：

```
$('.ui-selected').each(function(){
  $('<input>')
    .attr({
      type: 'hidden',
      name: 'title[]',
      value: $('td:first-child span',this).html()
    })
    .appendTo('#labForm');
});
```

如果想创建一个查询字符串来呈现`title[]`请求参数，那么可以编写如下代码：

```
var queryString = $.param({'title[]':
  $.map($('.ui-selected'),function(element){
    return $('td:first-child span',element).html();
  })
});
```

练习一，编写一些代码来获取当前选择的电影元素并通过 Ajax 请求`$.post()`来提交它们。

练习二，将之前在表单中创建隐藏字段的代码提取出来并创建为一个 jQuery 插件方法。

这就完成了对 jQuery UI 交互部件的探索。下面来回顾一下学到了哪些知识。

10.6 小结

本章中，我们继续探索 jQuery UI，专注于它提供的鼠标交互技术。

首先从拖动部件开始，它提供了一个基本的交互元素，这些元素为其他的交互部件运转的基石，其他那些部件包括可放置部件、可排序部件、可改变尺寸部件以及可选择部件。

我们看到了拖动交互部件如何使元素从页面布局中脱离开来，从而使得我们可以在页面中自由地移动它们。为了满足不同需求（也为了满足剩余交互部件的需求），拖动部件提供了很多选项可以用来自定义拖动的行为。

可放置部件提供了放置可拖动元素的有趣行为，从而提供了各种不同的用户界面语义动作。

有一种交互行为非常常见，因此 jQuery UI 为此提供了一个专门的交互部件——可排序部件。它允许我们拖动和放置元素来重新定义它们在有序列表（甚至是跨越多个不同的列表）中的位置。jQuery UI 不满足于只是让我们将元素移来移去，还提供了能够改变元素尺寸的可改变尺寸部件，这个部件拥有大量的选项来自定义调整尺寸的方式、方法。

最后，我们介绍了可选择部件，它可以向一个本身不具备可选择性的元素应用一个持久的选择状态。

总之，这些交互部件给予了我们强大的能力，可用来实现呈现给用户的、复杂的但易于使用的用户界面。

还没有完。这些交互部件也是 jQuery UI 提供的更加高级部件的基础。在下一章中，我们将继续探讨 jQuery UI，详细介绍它提供的用户界面部件。

第 11 章 jQuery UI 部件：超越 HTML 控件

本章内容

- 使用 jQuery UI 部件扩展 HTML 控件集
- 增强 HTML 按钮
- 为数字和日期的输入使用滑动条和日期选择器控件
- 直观地显示进度
- 使用自动完成部件来简化长的列表
- 使用选项卡部件和手风琴部件来组织内容
- 创建对话框

自从 Web 开始兴起以来，开发者们一直被 HTML 所提供的有限控件集所限制。虽然 HTML 控件集涵盖了从简单的文本输入到复杂的文件选择的所有内容，但是与桌面应用程序所使用的控件相比，这些控件还是黯然失色。HTML5 承诺要扩展这个控件集，但是要被所有主要的浏览器支持还需要一段时间。

例如，你是否经常听到 HTML 的`<select>`元素被称为“组合框”（对比桌面控件组合框，`<select>`元素只是行为上类似而已）？真正的组合框是非常有用的控件，经常出现在桌面应用中，然而 Web 开发者却无缘于组合框的优点①。

但是随着计算机更新换代，浏览器已经增强了其功能，DOM 操作已经成为了司空见惯的活动，聪明的 Web 开发者已经开始着手开发功能丰富的应用。通过创建扩展的控件（扩展现有的 HTML 控件或使用基本的元素从头创建控件），开发者社区已经展示了使用现有工具来创建丰富交互部件的创造力。

站在核心 jQuery 库的肩膀上，jQuery UI 为我们这些 jQuery 用户带来了创造力，通过提供一套自定义控件来解决常见的输入问题（很难使用传统的基本控件集来解决）。通过使标准的元素和其他元素协调一致地工作（视觉效果也很好），并接受一定区间内的数字值，来设置日期值或以新的方式来组织内容。jQuery UI 提供了一套有价值的部件集，为用户在页面上输入数据提供

① 桌面应用中的组合框功能强大，用户不仅可以从预先定义的列表中选择一项，也可以直接在文本框中输入，然而 HTML 的`<select>`元素只允许用户从预定义列表中选择一项。

更愉快的体验（同时也可以简化编码）。

随着对 jQuery UI 提供的核心交互部件的深入讨论，我们会继续观察 jQuery UI 都提供了哪些自定义控件（部件，这些控件给予我们更多选项来接受用户的输入）来弥补 HTML 控件集的不足。在本章中，我们将会探讨如下 jQuery UI 部件。

- 按钮（11.1 节）
- 滑动条（11.2 节）
- 进度条（11.3 节）
- 自动完成器（11.4 节）
- 日期选择器（11.5 节）
- 选项卡（11.6 节）
- 手风琴（11.7 节）
- 对话框（11.8 节）

和前一章一样，本章也很长！并且和交互部件一样，jQuery UI 方法使用一种独特的模式来创建部件从而使其易于理解。与交互部件不同的是，本章中的部件是相互独立的，因此你可以根据自己的喜好来阅读。

先从一个可以修改现有控件元素样式的简单部件开始：按钮。

11.1　按钮和按钮组

在我们为 HTML 4 控件集中控件种类的缺乏而感到惋惜的同时，HTML 4 控件集却提供了很多功能重复的按钮控件。

这就是`<button>`元素，以及不少于 6 种按钮语义的`<input>`元素：`button`、`submit`、`reset`、`image`、`checkbox` 以及 `radio`。此外，`<button>`元素拥有子类型 `button`、`submit` 以及 `reset`，它们的语义与那些相应的输入元素类型重复。

注意　为什么有那么多的 HTML 按钮类型？最初只有`<input>`按钮类型是可用的，但是由于它们只能用一个简单的文本字符串来定义，因此是有限制的。后来添加了`<button>`元素，它可以包含其他元素因此提供了更多的渲染方式。由于基本的`<input>`元素的各种类型是绝对不能废除的，所以最终我们看到了很多功能重复的按钮类型。

这些按钮类型提供各种不同的语义，它们在页面中非常有用。但是，当更深入探索 jQuery UI 部件集的时候，我们将会看到它们默认的可见样式与各种部件所展示的样式配合得不是很好。

11.1.1　UI主题中的按钮外观

还记得在第 9 章最开始我们下载了 jQuery UI 吗？有多个主题可供我们来选择下载，每一个主题都会向 jQuery UI 元素应用不同的外观。

为了让按钮匹配这些样式，我们可以在 CSS 文件中查找选择的主题，尝试找出可以应用到

按钮元素的样式，使按钮元素与其他元素的外观相协调。但是事实证明，我们不需要这么做——jQuery UI 提供了一个方法来增强按钮控件以便使其外观与主题相匹配，而无需改变按钮元素的语义。此外，jQuery UI 还给予了按钮元素悬停样式（当鼠标指针在按钮元素上悬停时，用来轻微改变其外观）——这是无样式按钮所缺少的功能。

button()方法会修改单个的按钮来增强其外观，而 buttonset()方法对一组按钮起作用（通常是一组单选按钮或复选框），不仅设置这些按钮的主题，而且会让它们看起来像是一个紧密结合的整体。

思考图 11-1 中显示的按钮。

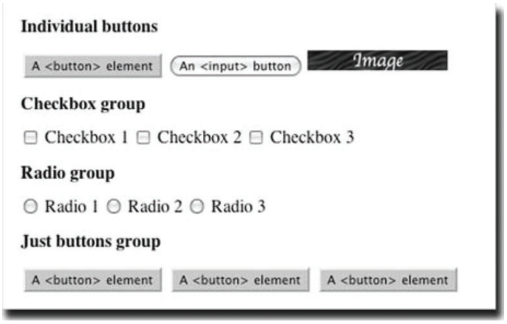


图 11-1　没有任何样式的各种按钮元素——相当单调，是吧

这个页面片段展示了一些没有应用主题的单个按钮元素、复选框分组、单选按钮以及<button>元素的外观。所有的按钮在功能上完全可以正常工作，但是在视觉上并不吸引人。

在给单个按钮应用了 button()方法，并给按钮分组应用了 buttonset()方法之后（使用 Cupertino 主题的页面），外观改变成图 11-2 所示的样子。

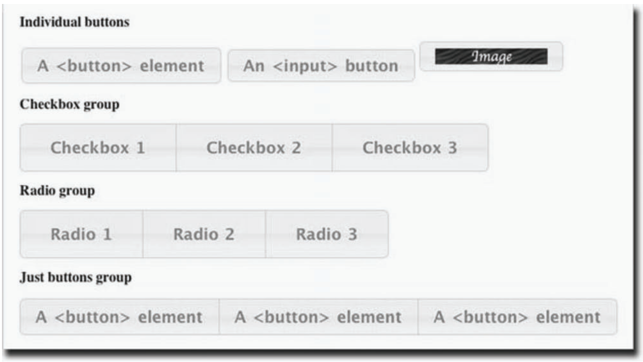


图 11-2　在进行了样式改造后，按钮都换上了最好看的样式并且准备好去吸引用户的注意力

在应用了样式之后，图 11-1 中显示的按钮看起来就相当地简朴了。

不仅仅是按钮的外观被改变以匹配主题，而且分组也应用了样式以便使分组中的按钮形成一个视觉整体来匹配它们的逻辑分组。尽管单选按钮和复选框已经被重设样式，从而看起来像“正

常的”按钮，但是它们仍然还保持有其语义上的行为。当引入 jQuery UI 按钮实验室后，我们将会在实际运行中看到上述操作。

随着在本章的剩余部分对 jQuery UI 部件的学习，我们将会越来越熟悉这个主题样式[①]。

不过首先来看一下将这个样式应用到按钮元素上的方法。

11.1.2　创建带有主题的按钮

jQuery UI 所提供的用来创建部件的方法与我们在前一章中看到的交互方法遵循相同的方式：首先调用 `button()`方法并传递一个散列选项来创建部件，然后再次调用相同的方法，传递一个字符串（用来标识以部件为目标的操作）来修改部件。

`button()`和 `buttonset()`方法的语法与我们研究的 UI 交互部件的方法语法类似，如下所示。

命令语法：button 和 buttonset

button(options)
button('disable')
button('enable')
button('destroy')
button('option', optionName, value)
buttonset(options)
buttonset('disable')
buttonset('enable')
buttonset('destroy')
buttonset('option', optionName, value)

为包装集中的元素设置主题以便使其匹配当前已加载的 jQuery UI 主题。甚至可以为非按钮元素（比如<span>和<div>）应用按钮外观和语义。

参数

options	（对象）一个散列对象，由要应用到包装集元素上的选项组成（参见表 11-1），从而使这些元素成为带有主题的按钮
'disable'	（字符串）禁用包装集中元素的单击事件
'enable'	（字符串）重新启用包装集中元素的按钮语义
'destroy'	（字符串）使元素恢复到应用 UI 主题之前的初始状态
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是 jQuery UI 按钮元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数

① 指的是 jQuery UI 的 Cupertino 主题。

optionName	（字符串）要设置或者返回的选项名称的值（参见表 11-1）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
value	（对象）要设置的选项的值（通过 optionName 参数来标识）

返回值

包装集，除了返回选项值的情况

为了将按钮主题应用到元素集上，我们使用一组选项来调用 button()或 buttonset()方法，或者不带参数接受默认的选项。下面是一个示例：

```
$(':button').button({ text: true });
```

注意　“设置主题”（theming）这个词语在字典里是没有，它是 jQuery UI 特有的，因此我们也使用它。

创建按钮时可用的选项如表 11-1 所示。

button()方法包含大量的选项，你可以在按钮实验室页面尝试使用它们，按钮实验室页面如图 11-3 所示，你也可在 chapter11/buttons/lab.buttons.html 中找到这个页面。



图 11-3　jQuery UI 按钮实验室页面可以让我们看到按钮在应用样式之前和之后的状态，还可以修改其中的选项

你可以一边阅读表 11-1 中的选项列表，一边在这个实验室做相应的练习。

表11-1　jQuery UI按钮和按钮组的选项

选　　项	描　　述	是否在实验室页面中
`icons`	（对象）指定一个或两个要在按钮中显示的图标：主要图标在左边，次要图标在右边。主要图标由对象的`primary`属性来标识，次要图标由`secondary`属性来标识 这些属性的值必须是174个被支持的调用名称之一（与jQuery按钮图标集相一致）。我们马上就会讨论到这些图标 如果省略，则不会显示图标	✓
`label`	（字符串）指定要在按钮上显示的、覆盖原始标签的文本。如果省略，则会显示元素的原始标签。对于单选按钮和复选框而言，原始标签就是与控件相关联的`<label>`元素	✓
`text`	（布尔）指定是否要在按钮上显示文本。如果指定为`false`，且`icons`选项至少指定了一个图标，则不显示文本。默认显示文本	✓

除了 `icons` 选项外，其他选项都很简单明了。下面就来简要探讨下 `icons` 选项。

11.1.3　按钮图标

jQuery UI 提供了一个可以显示在按钮上的、由 174 个主题图标组成的集合。你可以在左边显示单个图标（主要图标），也可以一个在左边一个在右边（次要图标）。

`icons` 是用来标识图标的类名。例如，要创建一个带有（小扳手图标的按钮，可以使用这段代码：

```
$('#wrenchButton').button({
  icons: { primary: 'ui-icon-wrench' }
});
```

如果想在左边显示星型图标，在右边显示心型图标，可以这么做：

```
$('#weirdButton').button({
  icons: { primary: 'ui-icon-star', secondary: 'ui-icon-heart' }
});
```

一张图片胜过千言万语，所以我们提供了一个包含所有按钮图标的页面（使用图标的名称来标明按钮），而不是在这里列出可用的图标名称。这个页面如图 11-4 所示，你也可以在 chapter11/buttons/ui-button-icons.html 找到它。

你可以保存这个页面，以便随时查找要在按钮上使用的图标。

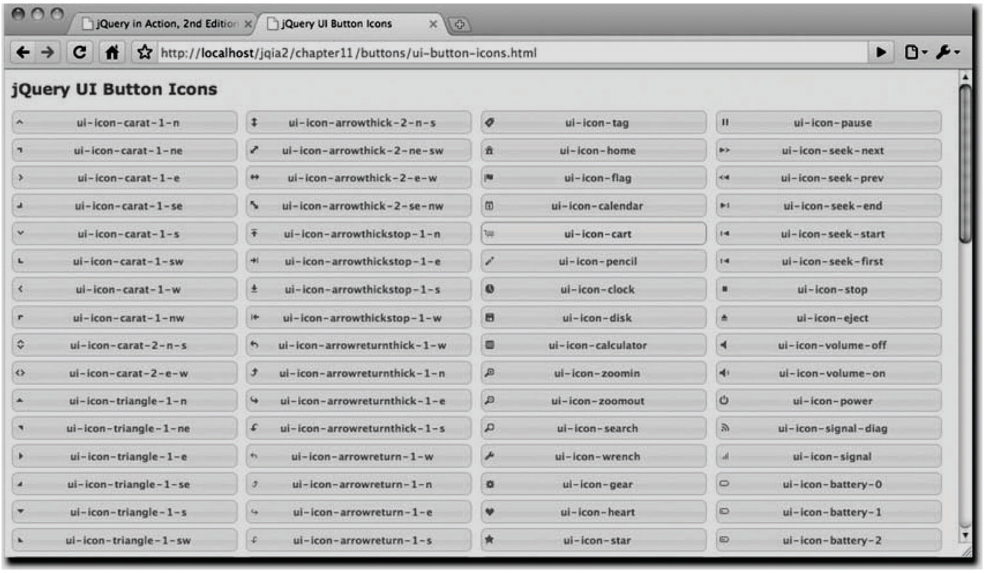


图 11-4 jQuery UI 按钮图标页面可以让我们看到所有可用的按钮图标及其名称

11.1.4 按钮事件

与交互部件及其他部件不同，jQuery UI 按钮没有关联的自定义事件。

因为这些部件仅仅是现有 HTML 4 控件的带主题版本，因此可以像没扩展之前一样使用原生事件。要处理按钮单击，只需要继续处理按钮的单击事件就可以了。

11.1.5 设置按钮样式

使用 jQuery UI 的 `button()`和 `buttonset()`方法的目的就是使按钮与所选择的 jQuery UI 主题相匹配。这就好像我们把按钮带上电视节目 What Not to Wear[①]（中译为“改头换面”，是美国和英国电视台的一档流行的换装真人秀节目）。这些按钮刚开始时很单调、丑陋，经过改造后却以惊艳的外观出现！但即便如此，我们也要调整这些元素的样式，使其在页面上更美观。例如，按钮图标页面上按钮的文本被调整为较小的尺寸以便适合页面上的按钮[②]。

jQuery UI 在创建部件时既扩展了已有元素又创建了新的元素，并且对匹配（定义在主题 CSS 样式表单中）样式规则的元素应用了类名。我们可以自行使用这些类名来扩展或覆盖在页面上的主题定义。

例如，在按钮图标页面，对按钮文本的字体大小进行了这样的调整：

```
.ui-button-text { font-size: 0.8em; }
```

① 这档节目每期都会选出一位他人推荐的参与者，他们的日常穿着均是十分怪异或没品味，急需改头换面。——编者注

② 这里指的是为 ui-button-icons.html 页面中按钮上的文本增加了 CSS 样式：`font-size:0.8em`。

类名 ui-button-text 被应用到包含按钮文本的<span>元素上。

覆盖（由 jQuery UI 创建的）部件的元素、选项以及类名的各个方面几乎是不可能的，因此我们没有去尝试。我们采取的方式是使用一些技巧，为每个部件类型应用一些可能需要在页面上使用的最常用的样式。前面讲过的改变按钮文本样式的技巧就是一个很好的示例。

按钮非常适合用户主动发起行动的情况，但是除了单选按钮和复选框，按钮并不能呈现我们可能想要从用户那里获取的值。一些代表逻辑表单控件的 jQuery UI 部件让我们能够轻松获取输入类型（而这个操作长期以来都是令人苦不堪言的）。下面就来看一个部件，它减轻了获取数字输入的负担。

11.2　滑动条

获取数字输入历来是 Web 开发者非常头痛的事情。HTML 4 控件集没有适合接受数字输入的控件。

文本字段可以（并且也经常）被用来接受数字输入。这并不是最优的方法，因为必须转换并验证用户输入的值，以确认用户没有输入“xyz”作为他们的年龄或居住的年数。

尽管事后验证的用户体验并不理想，但是限制文本控件只能输入数字也有问题。当用户持续单击 A 键但是没有任何事情发生时，用户可能会感到困惑。

在桌面应用中，要获取在一定区间内的数字值时通常会使用滑动条控件。与文本输入框相比，滑动条的优点是它不会让用户输入错误的值。用户用滑动条选择的任何值都是有效的值。

jQuery UI 提供了一个滑动条控件，这样我们就能利用它带来的好处了。

11.2.1　创建滑动条部件

滑动条通常表现为包含手柄的“水槽”形式。手柄可以沿着水槽移动以表示在该区间内选中的值，用户也可以在水槽中单击来表示手柄要移动到的地方。

滑动条可以是水平的，也可以是垂直的。图 11-5 展示了一个桌面应用程序中的水平滑动条。



图 11-5　滑动条可以用来表示一定区间内的值。在这个示例中，是从最小亮度到最大亮度的值

和 button()方法不同，滑动条不是通过扩展已有的 HTML 控件来创建的，而是使用基本的元素（例如<div>和<a>）组合而成的。设置目标<div>元素的样式来呈现滑动条的水槽，然后在其中创建锚点元素来形成手柄。

滑动条部件可以拥有任意数量的手柄，因此可以包含任意数量的值。使用数组来指定值，数组的每一项对应一个手柄。但是，由于单个手柄的情况比多个手柄的情况更为普遍，因此有专门的方法和选项来处理单个手柄的情况。

我们不用为只有单个手柄元素的情况处理数组，因为它是使用滑动条最常见的方式。感谢jQuery UI团队！我们非常感激你们所做的一切！

下面是slide()方法的语法。

命令语法：slider

slider(options)
slider('disable')
slider('enable')
slider('destroy')
slider('option', optionName, value)
slider('value', value)
slider('values', index, values)

把目标元素（推荐使用<div>元素）转变为滑动条控件

参数

options	（对象）一个散列对象，由要应用到包装集元素上的选项组成（参见表11-2），从而使元素成为滑动条
'disable'	（字符串）禁用滑动条控件
'enable'	（字符串）重新启用被禁用的滑动条控件
'destroy'	（字符串）使任何转变为滑动条控件的元素恢复到其初始状态
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是滑动条元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供optionName参数
optionName	（字符串）要设置或返回的选项名称的值（参见表11-2）。如果提供value参数，则这个值就成为选项的值。如果没有提供value参数，则返回已命名选项的值
value	（对象）和'option'一起使用时，指定要设置选项的值（通过optionName参数来标识）；和'value'一起使用时，指定为滑动条设置的值；和'values'一起使用时，指定为多个手柄设置的值
'value'	（字符串）如果提供value值，则为单手柄滑动条设置值并返回那个值；否则返回滑动条的当前值
'values'	（字符串）对于拥有多个手柄的滑动条，获取或者设置指定手柄的值，必须指定index参数来识别到底是哪个（或哪些）手柄。如果提供values参数，则设置手柄的值；否则返回指定手柄的值
index	（数字|数组）新值将要分配给的手柄下标或手柄索引的数组

返回值

包装集，除了返回选项值或手柄值的情况

在创建滑动条的时候，可以使用很多不同的选项来创建带有各种行为和外观的滑动条控件。

当你阅读表 11-2 中的选项时，请打开文件 chapter11/sliders/lab.sliders.html 中的滑动条实验室（如图 11-6 所示），然后跟着实验室进行练习，尝试不同的选项。



图 11-6　jQuery UI 滑动条实验室展示了可以用来建立和操作 jQuery UI 滑动条的各种方式

表11-2　jQuery UI滑动条的选项

选　　项	描　　述	是否在实验室页面中
`animate`	（布尔|字符串|数字）如果为`true`，则使手柄平稳地移动到在水槽内单击的位置。可以是一个持续时间值，也可以是如下字符串之一：`slow`、`normal`或`fast`。默认情况下，手柄会瞬间移动	✓

（续）

选　　项	描　　述	是否在实验室页面中
change	（函数）指定在滑动条上创建的函数，用来作为slidechange事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-3	✓
max	（数字）指定滑动条可以达到的区间内的上限值——该值表示了手柄可以移动到的（水平滑动条）最右端或者（垂直滑动条）顶端。默认该区间的上限值是100	✓
min	（数字）指定滑动条可以达到的区间内的下限值——该值表示了手柄可以移动到的（水平滑动条）最左端或者（垂直滑动条）底端。默认该区间的下限值是0	✓
Orientation	（字符串）horizontal 或vertical的其中之一。默认为horizontal	✓
range	（布尔|字符串）如果指定为true，则滑动条拥有两个手柄，并且在这两个手柄之间创建一个可以设置样式的元素。如果滑动条只有一个手柄，则指定min或max会分别创建一个从手柄到滑动条开始或结束位置的区间元素。默认不创建区间元素	✓
start	（函数）指定在滑动条上创建的函数，用来作为slidestart事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-3	✓
slide	（函数）指定在滑动条上创建的函数，用来作为slide事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-3	✓
step	（数字）指定滑动条可以在允许的最大值和最小值之间移动的最小步长值。例如，步进值为2只允许选择偶数数字值 步进值将会均匀地分割区间 默认情况下，step为1以便可以选择所有的值	✓
stop	（函数）指定在滑动条上创建的函数，用来作为slidestop事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-3	✓
value	（数字）指定单个手柄滑动条的初始值。如果有多个手柄（参见values选项），则指定第一个手柄的值 如果省略，则初始值就是滑动条的mini值	✓
values	（数组）创建多个手柄并指定这些手柄的初始值。这个选项应当是一个由可能的值组成的数组，每个值对应一个手柄 例如，[10,20,30]将会使滑动条带有三个初始值分别为10、20和30的手柄。 如果省略，则只创建一个手柄	✓

现在就来探讨滑动条控件可以触发的事件吧。

11.2.2　滑动条事件

和交互部件一样，当有趣的事情发生时大部分 jQuery UI 部件会触发自定义事件。我们可以使用两种方式其中之一来为这些事件创建处理器：可以在祖先层次结构中的任意位置以通常的方式来绑定处理器，也可以将处理器指定为一个选项（我们曾在上一节中看到过这种方式）。

例如，我们可能想要在 body 元素上以全局方式处理滑动条的滑动事件：

```
$('body').bind('slide',function(event,info){ ... });
```

我们可以使用单个的处理器为页面上的所有滑动条处理滑动事件。如果处理器是特定于一个滑动条实例的，那么当创建滑动条时就要改用 slide 选项：

```
$('#slider').slider({ slide: function(event,info){ ... } });
```

我们可以利用这种灵活性，以最适合页面的方式来创建处理器。

和交互部件的事件一样，每个事件处理器都会被传入两个参数：一个是事件示例，另一个是包含该控件信息的自定义对象。是不是非常像？

该自定义对象包含如下属性。

- handle——对被移动手柄<a>元素的引用。
- value——代表被移动手柄的当前值。对于单个手柄的滑动条来说，该属性被认为是滑动条的值。
- values——由滑动条所有手柄的当前值组成的数组，只有多个手柄的滑动条才有该属性。

在滑动条实验室，value 和 values 属性用来实时更新显示在滑动条下面的值。

表 11-3 总结了滑动条可以触发的事件。

表11-3　jQuery UI滑动条的事件

事　件	选　项	描　述
slide	slide	在滑动条中拖动手柄时，为mousemove事件触发。返回false会取消滑动
slidechange	change	当滑动条的值改变时会触发，通过用户活动或程序来改变滑动条的值
slidestart	start	当滑动开始时触发
slidestop	stop	当滑动停止时触发

slidechange 事件可能是最有趣的事件之一，因为它可以用来跟踪滑动条的单个值（或多个值）。

假设我们有一个单个手柄的滑动条，需要使用表单提交来将其值提交到服务器。再假设有一个名为 sliderValue 的隐藏输入框用来保存滑动条的最新值，以便在外面表单提交时，滑动条的值表现得就像是另一个表单控件一样。我们可以在表单上创建如下事件：

```
$('form').bind('slidechange',function(event,info){
  $('[name="sliderValue"]').val(info.value);
});
```

下面是一个帮助你理解的快速练习。

- **练习 1**——只要表单中只有一个滑动条，那么前面的代码就可以很好地工作。修改前面的代码使其可以在包含多个滑动条的情况下正常工作。如何才能识别与单个滑动条控件相对应的隐藏输入元素呢？

现在来给滑动条添加一些样式。

11.2.3　为滑动条添加样式的技巧

当将元素转变为滑动条时，就会向其添加类 ui-slider。在该元素中，将会创建<a>元素来表示手柄，每个<a>元素都会被赋予类名 ui-slider-handle。我们可以使用这些类名来扩展所选择元素的样式。

提示　你能猜猜为什么要使用锚点元素来表现手柄吗？时间到——这是因为手柄是可聚焦的元素。在滑动条实验室中，创建滑动条并通过单击手柄来聚焦此手柄。现在使用左箭头按键和右箭头按键来看看会发生什么。

另一个将要被添加到滑动条元素上的类是 ui-slider-horizontal 和 ui-slider-vertical 两者之一（取决于滑动条的方向）。这是个很有用的钩子（hook），我们可以基于滑动条的方向来调整其样式。例如，在滑动条实验室你可以找到如下的样式规则，用来根据滑动条的方向来调整其尺寸：

```
.testSubject.ui-slider-horizontal {
  width: 320px;
  height: 8px;
}
.testSubject.ui-slider-vertical {
  height: 108px;
  width: 8px;
}
```

testSubject 是在实验室中用来识别要转变为滑动条元素的类名。

还有另一个优雅的技巧：假设为了和站点其余元素的样式相匹配，我们想让滑动条手柄看起来像一个鸢尾花。用一张合适的图片和一个 CSS 的小戏法，我们就能实现这个效果。

打开滑动条实验室，重置所有元素，选中标签为 Use Image Handle 的复选框，然后单击 Apply。滑动条看起来如图 11-7 所示。



图 11-7　用一张 PNG 图片和一个 CSS 小戏法，就能让滑动条手柄看起来像任何想要的效果

下面来讲解如何实现这个目标。首先，创建一张背景透明并且包含鸢尾花的 PNG 图片，将其命名为 handle.png，图片大小为 18 像素 × 18 像素。然后将下面的样式规则添加到页面中：

```
.testSubject a.ui-slider-handle.fancy {
  background: transparent url('handle.png') no-repeat 0 0;
  border-width: 0;
}
```

最后，在滑动块创建之后，将 fancy 类添加到手柄。

```
$('.testSubject .ui-slider-handle').addClass('fancy');
```

最后一个技巧：如果你通过 range 选项来创建区间元素，那么可以使用 ui-widget-header 类来为其添加样式。我们在实验室页面中是使用下面这行代码来实现的：

```
.ui-slider .ui-widget-header { background-color: orange; }
```

滑动条是一种很棒的输入工具，可以让用户输入一定区间内的数字值，而且还减轻了我们和用户的工作量。下面来看另一个可以提升用户体验的插件。

11.3　进度条

对用户来说一种非常不友好的体验是，耐着性子等待一个长时间的操作，还不知道后台是否真正有事情发生。虽然相对于桌面应用程序而言，用户在某种程度上更习惯于等待 Web 应用程序的处理结果，但是给用户反馈，告诉他们数据正在被处理，会让用户更加高兴，避免他们对此忧心忡忡。

这对于我们的应用程序来说也是有好处的。让用户一边在界面上疯狂单击鼠标，一边对着电脑屏幕嚷嚷“我的数据怎么还没出来！”，那后果可是不堪设想的。由此产生的错乱请求会使服务器瘫痪，最坏的情况可能导致后台代码产生问题。

进度条是一个非常棒的工具，它能够以相当准确且确定性的方式来告知用户操作所完成百分比，通知他们后台正在有事情发生。

当没有使用进度条的时候

比猜测操作完成的时间更让用户恼火的是：向用户谎报操作的进度。

只有能够基本保证汇报的准确度时，才应该使用进度条。如果进度条在达到了 10%的时候突然跳到终点（会导致用户认为操作可能中途被终止了），或者更糟的是——早在操作实际完成前进度条就显示进度为 100%，那么使用进度条就不是一个好方法。

如果你不能确定准确的已完成百分比，那么代替进度条的一个好方法是，给出某种提示来告知有些操作可能要花很长时间来完成；这个提示可以是一个文本显示“请等待，数据正在处理中——此过程可能需要几分钟……”，也可以用一个动画来在进行长时间操作的时候给用户以正在运行的错觉。

对于后一种替代方案，你可以访问一个方便生成 GIF 动画的网站：http://www.ajaxload.info/，然后再调整生成的动画来匹配你的主题。

进度条在外观上通常是矩形，通过从左向右逐渐“填充”一个不同外观的内部矩形来表示操作的完成进度。图 11-8 所展示的进度条示例表现的是一个完成进度略少于一半的操作。



图 11-8　通过从左向右“填充”控件来展示操作完成百分比的进度条

JQuery UI 提供了一个易用的进度条部件，用来让用户知道应用程序正在执行所请求的操作。下面来看下使用进度条究竟有多么容易。

11.3.1　创建进度条

意料之中的是，进度条使用 progressbar()方法来创建，该方法同样遵循我们早已熟悉的模式。

命令语法：progressbar

progressbar(options)
progressbar('disable')
progressbar('enable')
progressbar('destroy')
progressbar('option', optionName, value)
progressbar('value', value)

将包装元素（推荐使用<div>元素）转变为进度条部件

参数

options	（对象）一个散列对象，由要应用到所创建进度条的选项组成（参见表 11-4）
'disable'	（字符串）禁用进度条
'enable'	（字符串）重新启用被禁用的进度条
'destroy'	（字符串）将已转变为进度条部件的元素恢复到其初始状态
'option'	（字符串）基于剩余的参数，在包装集的所有元素上设置选项值，或者从包装集中的第一个元素（必须是进度条元素）获取选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
optionName	（字符串）选项的名称（参见表 11-4），该选项的值将被设置或者返回。如果提供 value 参数，则这个值会成为选项的值。如果没有提供 value 参数，则已命名选项的值会被返回
value	（字符串|数字）和'option'一起使用时，指定要设置选项的值（通过 optionName 参数来标识）;和'value'一起使用时,指定为滑动条设置的 0～100 之间的值
'value'	（字符串）如果提供 value 值，则为进度条设置该值；否则返回进度条的当前值

返回值

包装集，除了返回选项值的情况

进度条是一个很简单的部件，而且这个简单性也反映在 progressbar()方法可用的选项列表中。可用的选项只有两个，如表 11-4 所示。

表11-4　jQuery UI进度条的选项

选　　项	描　　述
change	（函数）指定在进度条上创建的函数，用来作为progressbarchange事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-5
value	（数字）指定进度条的初始值；如果省略，则默认为0

创建了一个进度条后，更新该进度条的值就很简单，只需调用该方法的 value 变体即可：

```
$('#myProgressbar').progressbar('value',75);
```

Value 的值只能设置为 0 ~ 100 之间的值，大于 100 的值会被设置为 100，负值会被设置为 0。

这些选项很简单，就和为进度条所定义的事件一样。

11.3.2　进度条事件

为进度条定义的单个事件如表 11-5 所示。

需要捕获 progressbarchange 事件，以便实时、准确地显示控件完成的百分比，或者出于任何其他原因需要通知页面什么时候值发生了改变。

表11-5　jQuery UI进度条的事件

事　　件	选　　项	描　　述
progressbarchange	change	每当进度条的值改变时调用。向其传递两个参数：事件实例和一个空对象。传入后者是为了与其他的jQuery UI事件保持一致，但是该对象内不包含任何信息

进度条非常简单（只有两个选项和一个事件），因此没有为该控件提供实验室页面。相反，我们认为应该创建一个插件来在长时间操作进行的时候动态地更新进度条。

11.3.3　自动更新的进度条插件

当触发了一个需要长时间处理（处理时间可能超出常人可接受的忍耐限度）的 Ajax 请求，而且我们可以准确地获取完成百分比时，通过显示进度条来取悦用户是一个好主意。

下面来考虑下完成该任务所需要进行的步骤。

(1) 触发一个长时间的 Ajax 操作。

(2) 创建一个默认值为 0 的进度条。

(3) 每隔一段时间，触发额外的请求来获取长时间操作的脉冲并返回完成的状态。这个操作必须是快速和准确的。

(4) 使用返回的结果来更新进度条和显示完成百分比的文本。

听起来相当容易，但是还有一些细节需要考虑。例如，确保在正确的时间销毁间隔计时器。

1. 定义自动更新进度条部件

由于这个部件是很常用的，并且还有一些重要细节需要考虑在内，因此非常适合创建一个用来处理上述事情的插件。

我们称该插件为自动更新进度条，其方法 autoProgressbar()的定义如下所示。

命令语法：autoProgressbar

autoProgressbar(options)

autoProgressbar('stop')

autoProgressbar('destroy')

autoprogressbar('value', value)

将包装元素（推荐使用<div>元素）转变为进度条部件

参数

options	（对象）一个散列对象，由要应用到所创建进度条的选项组成（参见表 11-6）
'stop'	（字符串）停止检查自动更新进度条部件的完成状态
'destroy'	（字符串）停止自动更新进度条部件并恢复已转变为进度条部件的元素的初始状态
'value'	（字符串）如果提供 value，则为进度条设置该值；否则返回进度条的当前值
value	（字符串|数字）和'value'一起使用时，指定为滑动条设置的 0～100 之间的值

返回值

包装集，除了返回选项值的情况

为该插件定义的选项如表 11-6 所示。

表11-6 autoProgressbar()插件方法的选项

选　项	描　述
pulseUrl	（字符串）指定服务器端资源的URL，用来检查希望监听的后端操作的脉冲。如果省略该选项，则此方法不执行操作 来自于该资源的响应必须由范围在0～100的数字值表示，用来表示被监听操作的完成百分比
pulseData	（对象）应当传入由pulseUrl选项所获取的任何数据。如果省略，则不发送数据
interval	（数值）两次脉冲之间的间隔时间（以毫秒为单位）。默认值是1000（1秒）
change	（函数）要创建为progressbarchange事件处理器的函数

下面就来实现这些内容。

2. 创建自动更新进度条

与往常一样，我们将会从该方法的架构入手，这个架构遵循在第 7 章中阐述的规则和实践。（如果下面的代码看起来有点生疏，请复习第 7 章。）在文件 jquery.jqia2.autoprogressbar.js 中编写了如下架构代码：

```
(function($){
  $.fn.autoProgressbar = function(settings,value) {

//在此放置实现主体

    return this;
  };

})(jQuery);
```

我们要做的第一件事情是，检查第一个参数是否为字符串。如果第一个参数是字符串，则使用该字符串来决定要运行的方法。如果不是字符串，则假定它是一个散列选项。因此我们添加了如下的条件结构：

```
if (typeof settings === "string") {
  // 在此处理方法
}
else {
 // 在此处理选项
}
```

处理选项是自动更新进度条插件的主要内容，因此我们先从处理 `else` 部分的代码着手。首先，将用户提供的选项和默认的选项集合并，如下所示：

```
settings = $.extend({
  pulseUrl: null,
  pulseData: null,
  interval: 1000,
  change: null
},settings||{});
if (settings.pulseUrl == null) return this;
```

和前面已开发的插件一样，我们使用`$.extend()`函数来合并不同的对象。还需要注意的是，我们习惯性地继续列出默认散列中的所有选项，即使该选项的值为 `null`。这是一个理想的地方来观察插件支持的所有选项。

在合并以后，如果没有指定 `pulseUrl` 选项[①]，则直接返回不进行任何操作（如果不知道如何访问服务器的话，也做不了什么事情）。

现在真正是创建进度条部件的时候了：

```
this.progressbar({value:0,change:settings.change});
```

请记住，在插件内，`this` 是对包装集的引用。我们在这个包装集上调用 jQuery UI 进度条方法，指定初始值为 `0`，并且传入用户提供的任何改变处理器。

① 这个代码不严谨，如果指定 `pulseUrl` 为 `undefined` 或空字符串的话，需要做相同的处理；因此这段代码最好这么写：`if(!settings.pulseUrl) return this;`。

现在来看有趣的部分。对于包装集中的每个元素（包装集中有可能只有一个元素，但是为什么要给自己添加限制呢？），我们希望使用已提供的 pulseUrl 来启动一个间隔计时器，以便检查长时间操作的状态。用来实现该任务的代码如下所示：

```
this.each(function(){
  var bar$ = $(this);
  bar$.data(
    'autoProgressbar-interval',
    window.setInterval(function(){
      $.ajax({
        url: settings.pulseUrl,
        data: settings.pulseData,
        global: false,
        dataType: 'json',
        success: function(value){
          if (value != null) bar$.autoProgressbar('value',value);
          if (value == 100) bar$.autoProgressbar('stop');
        }
      });
    },settings.interval));
});
```

❶ 遍历包装集
❷ 在部件上保存间隔计时器句柄
❸ 启动间隔计时器
❹ 触发 Ajax 请求
❺ 接收完成状态

这段代码处理的事情很多，因此我们依次讲解每一个步骤。

我们想让每个将被创建的进度条都拥有自己的间隔计时器。为什么用户会想要创建多个自动更新进度条超出了我们的职责范围，但 jQuery 的方式是让每个包装元素都拥有自己的操作范围。我们使用 each()方法❶来单独处理每个包装元素。

为了可读性，也为了能够在稍后创建的闭包中使用，我们将包装元素保存在 bar$变量中。

然后启动间隔计时器，但是不要忘了稍后停止计时器。因此需要将识别计时器的句柄存储到稍后容易获取的地方。jQuery 的 data()方法在这里就派上用场了❷，我们使用该方法来将计时器的句柄存储到 bar 元素上（存储的名称为 autoProgressbar-interval）。

调用 JavaScript 的 window.setInterval()函数启动该计时器❸。向这个函数传递一个要在计时器每次计时的时候执行的内联函数，以及从 interval 选项获取的时间间隔值。

在计时器回调内部，我们向 pulseUrl 选项所提供的 URL 发起（带有 pulseData 提供的数据）Ajax 请求❹。我们也关闭了全局事件（这些请求是在幕后发生的，我们不希望由于触发页面一无所知的全局 Ajax 事件使页面混乱不堪），并指定返回的响应体为 JSON 数据。

最后，在请求的 success 回调函数中❺，我们使用完成百分比更新进度条（此百分比作为响应体返回并被传入回调）。如果该值达到 100，表明操作已经完成，我们通过调用 stop 方法来停止计时器。

在此之后，实现剩余方法就会很简单。在高层次条件语句的 if 部分（用来检查第一个参数是否为字符串的那部分代码），编写如下代码：

```
switch (settings) {
  case 'stop':
    this.each(function(){
```

❶ 对字符串值进行条件切换
❷ 实现 **stop** 方法

```
      window.clearInterval($(this).data('autoProgressbar-interval'))
    });
    break;
  case 'value':
    if (value == null) return this.progressbar('value');
    this.progressbar('value',value);
    break;
  case 'destroy':
    this.autoProgressbar('stop');
    this.progressbar('destroy');
    break;
  default:
    break;
}
```

实现 **value** 方法

实现 **destroy** 方法

对于不支持的字符串什么都不做

在这个代码片段中，我们根据 `settings` 参数中的字符串切换到不同的处理算法❶，包括 `stop`、`value` 或 `destroy` 其中之一。

对于 `stop` 方法，我们希望消除为包装集中元素创建的所有间隔计时器❷。我们获取计时器句柄（此句柄被存储为元素的数据），并把它传入 `window.clearInterval()` 方法来停止计时器。

如果指定方法为 `value`，则将此值传入进度条部件的 `value` 方法。

当指定为 `destroy` 时，我们希望停止计时器，因此调用自身的 `stop` 方法（为什么要复制粘贴相同的代码两次呢？），然后销毁进度条。

这样就大功告成了！注意每当从方法的调用返回时，我们是如何返回包装集以便插件可以参与到 jQuery 方法链的（就像任何其他的可链式调用的方法一样）。

这个插件完整的实现可以在文件 chapter11/progressbars/jquery.jqia2.autoprogressbar.js 中找到。

下面把注意力转向如何测试我们的插件。

3. 测试自动更新进度条插件

文件 chapter11/progressbars/autoprogressbar-test.html 包含了一个使用新插件的测试页面，用来监听一个长时间运行的 Ajax 操作的完成进度。为了节省篇幅，我们不会列出那个文件中的每一行代码，而是重点研究插件相关的部分代码。

首先，来看下创建 DOM 结构的标签：

```
<div>
  <button type="button" id="startButton" class="green90x24">Start</button>
  (starts a lengthy operation)
</div>

<div>
  <div id="progressBar"></div>
  <span id="valueDisplay">&mdash;</span>
</div>

<div>
  <button type="button" id="stopButton" class="green90x24">Stop</button>
  (stops the progress bar pulse checking)
</div>
```

这些标签创建了 4 个基本元素：

- ❑ 一个用来启用长时间操作的 Start 按钮，并使用我们的插件来监听它的进度；
- ❑ 一个将要被转变为进度条的<div>；
- ❑ 一个用来显示完成百分比文本的<span>；
- ❑ 一个用来停止进度条监听长时间操作的 Stop 按钮。

测试页面运行中的效果如图 11-9 所示。

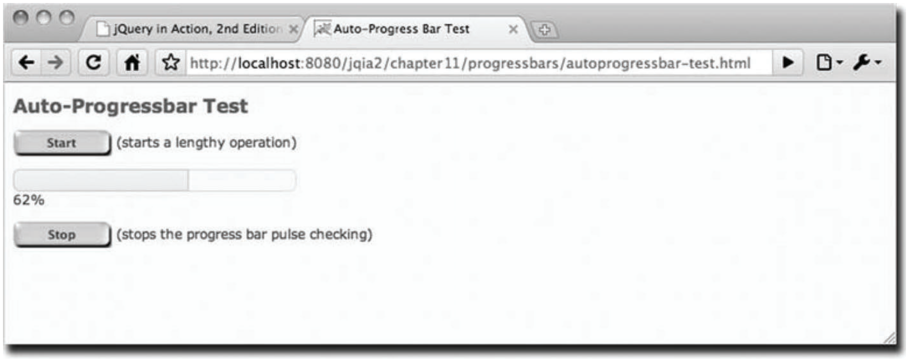


图 11-9　自动更新进度条正在监听服务器端长时间运行的操作

注意　因为这个示例使用了服务器端 Ajax 操作，因此它必须运行在（第 8 章为示例创建的）Tomcat 实例下（注意 URL 中的 8080 端口）。你可以访问 http://www.bibeault.org/jqia2/chapter11/progressbars/autoprogressbar-test.html 页面来远程运行此示例。

为 Start 按钮添加行为是本页面中最重要的操作，这是通过如下脚本实现的：

```
$('#startButton').click(function(){
  $.post('/jqia2/lengthyOperation',function(){
    $('#progressBar')
     .autoProgressbar('stop')
     .autoProgressbar('value',100);
  });
  $('#progressBar').autoProgressbar({
    pulseUrl: '/jqia2/checkProgress',
    change: function(event) {
      $('#valueDisplay').text$('#progressBar').autoProgressbar ('value') +
     '%');
    }
  });
});
```

❶ 开始长时间运行的处理

❷ 使操作结束

❸ 创建监听进度条

在 Start 按钮的单击处理器中，我们做了两件事情：开始长时间操作和创建自定更新进度条。

我们向 URL /jqia2/lengthyOperation 发起请求，这个 URL 是用来标识在服务器端运行的、大约需要 12 秒来完成的处理❶。稍后会谈到 success 回调函数，不过首先来看看自动更新进度条的创建过程。

我们使用用来标识服务器端资源的值（/jqia2/checkProgress）来调用新插件❸，这个资源用来检查长时间运行操作的状态并返回完成百分比作为响应。如何完成上述操作完全取决于 Web 应用程序后端代码，而这个话题超出了本次讨论的范围。（在本示例中，使用的是两个独立的小服务程序（servlet），使用服务程序会话来持续跟踪进度。）为进度条准备的改变处理器被用来更新屏幕上显示的完成值。

现在回到为长时间运行操作所准备的成功处理器❷。当操作完成的时候，我们希望做两件事情：停止进度条，确保进度条的值反映了操作已经全部完成。首先调用插件的 `stop` 方法，紧跟着调用 `value` 方法，就可以很容易地完成这个任务。进度条的改变处理器也会相应地更新文本值的显示。

我们已经使用进度条创建了一个真实可用的插件。下面就来讨论一些为进度条添加样式的技巧。

11.3.4　为进度条添加样式

如果把一个元素转变为进度条，会向其添加类名 `ui-progressbar`，并向此元素内用来描述进度条值的`<div>`元素添加类名 `ui-progressbar-value`。可以使用这些类名来应用 CSS 规则，从而扩展这些元素的样式（我们认为合适的样式）。

例如，你可能想使用一个有趣的图案而不是主题采用的纯色调来填充内部元素的背景：

```
.ui-progressbar-value {
  background-image: url(interesting-pattern.png);
}
```

你也可以提供一个动画 GIF 作为背景图片，使得进度条看起来更加生动。

进度条可以让用户了解操作处理进度，从而使其心情平静。下一步，我们通过让用户在查找想要的东西时减少用户的输入来使他们高兴。

11.4　自动完成部件

缩写词 TMI 现在代表“太多信息”（too much information），在谈话中它经常被用来指说话者披露了对听众来说有点过于私密的细节。在 Web 应用领域，“太多信息”指的不是信息的性质，而是信息的数量。

尽管 Web 上有大量可用信息是一件好事，但是信息真的可能会太泛滥——在大量数据面前我们很容易就会不堪重负。另外一个描述这种现象的常用说法是“从消防水龙头喝水①”。

在设计用户界面时，特别是为有能力获取大量数据的 Web 应用程序设计用户界面，一个重要的原则是避免向用户提供太多的数据或者太多的选择。当呈现庞大的数据集时（例如报表数据），好的用户界面会给予用户一些工具，以便能获取有用且有益的数据。例如，过滤器可以被用来筛选掉和用户无关的数据，对庞大的数据集进行分页以便以易于查看的块来呈现数据。这也

① 意指信息量太多而无法处理。

正是 DVD Ambassador 示例所采取的方式。

作为示例，下面来考虑一个将会在本节使用的数据集：一个 DVD 标题的列表，这是一个包含 937 个标题的数据集。它是一个庞大的数据集，但仍然是更加庞大的数据集（例如曾经发行过的所有 DVD 列表）的一小部分。

假设我们希望向用户呈现这个列表，让他们选择喜欢的 DVD。可以创建一个 HTML `<select>`元素让用户从中选择一个标题，但这对用户来说不太友好。大部分的可用性指南都推荐每次向用户呈现的选择不要超过一打，更不用说数百个选择了！先撇开可用性问题不谈，每次都向（可能有潜在的数百、数千，甚至数以百万的用户访问的）页面发送如此庞大的数据集是否现实？

jQuery UI 使用自动完成部件来帮助我们解决这个问题，这个控件和`<select>`下拉列表的大部分行为类似，但是它会过滤选择项，只呈现匹配用户输入控件关键字的那些选项。

11.4.1　创建自动完成部件

jQuery 的自动完成部件增强了现有的`<input>`文本元素，用来获取并呈现包含可能选择项的菜单，以匹配用户在文本字段中输入的任意文本。匹配选择项的内容取决于创建部件时所提供的选项。事实上，自动完成部件可以提供灵活的选择列表，还可以基于用户提供的数据筛选这个列表。

`autocomplete()`方法的语法如下所示。

命令语法：autocomplete

`autocomplete(options)`
`autocomplete('disable')`
`autocomplete('enable')`
`autocomplete('destroy')`
`autocomplete('option', optionName, value)`
`autocomplete('search', value)`
`autocomplete('close')`
`autocomplete('widget')`

将包装集中的`<input>`元素转变为自动完成控件

参数

`options`	（对象）一个散列对象，由要应用到包装集中元素的选项组成（参见表 11-7），从而使这些元素成为自动完成控件
`'disable'`	（字符串）禁用自动完成控件
`'enable'`	（字符串）重新启用被禁用的自动完成控件
`'destroy'`	（字符串）使任何转变为自动完成控件的元素恢复到其初始状态

'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是自动完成元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
optionName	（字符串）要设置或返回的选项名称的值（参见表 11-7）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
value	（对象）和 'option' 一起使用时，指定要设置选项的值（通过 optionName 参数来标识）；和 'search' 一起使用时，指定为要搜索的术语
'search'	（字符串）使用指定的 value（如果 value 值存在的话，否则使用控件当前的内容作为 value 的值）触发一个搜索事件。提供空字符串来查看所有可能项的菜单[①]
'close'	（字符串）如果自动完成菜单处于打开状态，则关闭此菜单
'widget'	（字符串）返回自动完成元素（也即是带有 ui-autocomplete 类名的元素）

返回值

包装集，除了返回选项、元素或者搜索结果[②]的情况

对于这样一个看起来很复杂的自动完成控件，可用的选项列表却相当少，如表 11-7 所述。

表11-7　jQuery UI自动完成部件的选项

选　项	描　述	是否在实验室页面中
change	（函数）指定在自动完成部件上创建的函数，用来作为autocompletechange事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-8	✓
close	（函数）指定在自动完成部件上创建的函数，用来作为autocompleteclose事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-8	✓
delay	（数值）在尝试获取匹配的值（由source选项指定）之前等待的毫秒数。它可以在发起搜索之前给用户输入更多的字符的时间，以便在获取非本地数据时减少对服务器的压力 如果省略，则默认是300（也即是0.3秒）	✓
disabled	（布尔）如果指定为true，则部件最初是被禁用的	
focus	（函数）指定在自动完成部件上创建的函数，用来作为autocompletefocus事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-8	✓

① 这个描述不严谨，只有在创建自动完成部件时指定选项 minLength 值为 0（默认为 1），才能使用 autocomplete('search', '')来显示所有项的菜单。

② 原文错误，指定 autocomplete 方法的第一个参数为'search'时，返回的结果仍然是包装集，而不是搜索结果。

（续）

选　项	描　述	是否在实验室页面中
minLength	（数值）在尝试获取匹配的值（由source选项指定）之前必须输入的字符数。它可以在提供的字符不足以将集合消减到合理的水平时，防止呈现过大的集合 默认值是1个字符	✓
open	（函数）指定在自动完成部件上创建的函数，用来作为autocompleteopen事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-8	✓
search	（函数）指定在自动完成部件上创建的函数，用来作为autocompletesearch事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-8	✓
select	（函数）指定在自动完成部件上创建的函数，用来作为autocompleteselect事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-8	✓
source	（字符串|数组|函数）指定获取匹配输入数据的方式。必须为此选项提供一个值，否则不会创建自动完成部件。这个值可以是返回匹配数据的服务器资源URL的字符串，提供匹配值的本地数据的数组，也可以是提供匹配值的通用回调函数 有关这个选项的更多信息请参阅11.4.2节	✓

你可能已经猜到了，我们提供了自动完成部件实验室页面（参见图 11-10）。请在浏览器中打开文件 chapter11/autocompleters/lab.autocompleters.html，并在复习选项的时候跟着做练习。

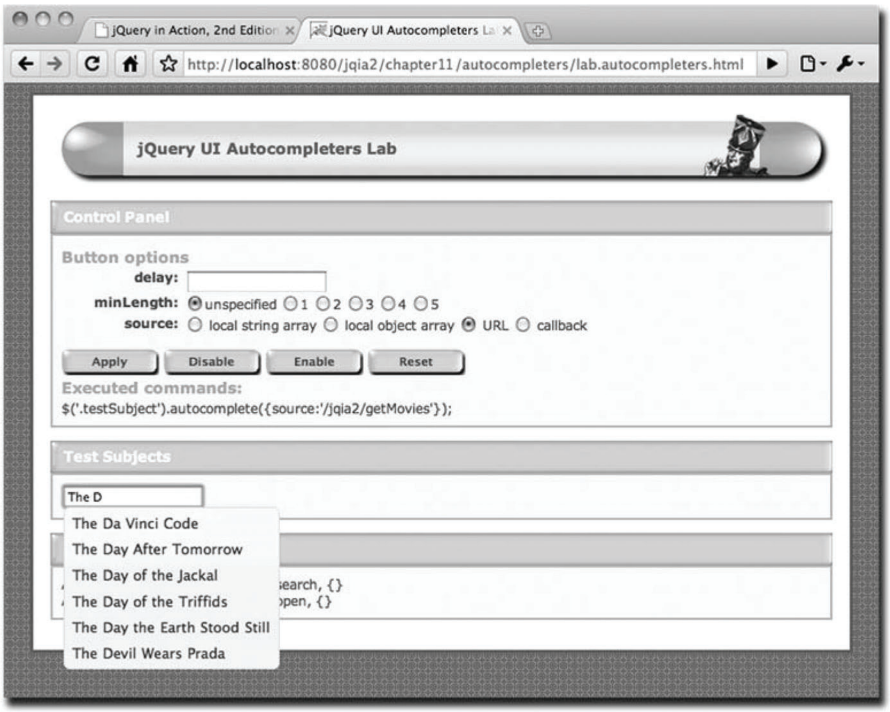


图 11-10　jQuery UI 自动完成实验室展示了如何随着输入数据的增多而减少大型结果集的数量

11

注意　在这个实验室中，source 选项的 URL 变体要求使用服务器端 Ajax 操作。它必须在（第 8 章为示例创建的）Tomcat 实例下运行（注意 URL 中的 8080 端口）。你也可以访问 http://www.bibeault.org/jqia2/chapter11/autocompleters/lab.autocompleters.html 页面来远程运行此示例。

除了 source 选项，其余的选项都还算是不言自明的。请保持 source 选项的默认设置，使用自动完成实验室页面来观察所生成的事件以及 minLength 和 delay 选项的行为，理解这些选项的用法。

下面来看看如何才能为这个部件提供数据源。

11.4.2　自动完成部件的数据源

自动完成部件为我们提供了很大的灵活性来为用户输入的任何数据提供匹配的数据值。

自动完成部件的数据源表现为数组的形式，数组中的每一项有两个属性，如下所示。

- 一个用来呈现真实值的 value 属性。它们是和用户输入控件的值进行匹配的字符串[①]，并且当一个菜单项被选中时它们的值被插入到控件中。
- 一个用来呈现值的 label 属性，通常以缩略的形式表示。它们是显示在自动完成菜单中的字符串，并且不参与默认的匹配算法[②]。

这些数据可以来自各种不同的源头。

对于数据集相当小的情况（一打，不超过一百个），可以将数据作为局部数组来提供。下面的示例来自于自动完成实验室，它提供了使用用户名作为标签、完整的名称作为值的候选数据。

```
var sourceObjects = [
  { label: 'bear', value: 'Bear Bibeault'},
  { label: 'yehuda', value: 'Yehuda Katz'},
  { label: 'genius', value: 'Albert Einstein'},
  { label: 'honcho', value: 'Pointy-haired Boss'},
  { label: 'comedian', value: 'Charlie Chaplin'}
];
```

在显示的时候，标签（也就是用户名）是显示在自动完成菜单中的值，但匹配是在值（也就是完整的名称）上进行的[③]，并且在选择某个菜单项的时候，这个值会被插入到控件中。

---

① 作者在此犯了一个严重的错误，以至于后面对 label 和 value 的描述都不完全正确。

正确的说法是：label 会和用户输入的值进行匹配并显示在菜单中；当聚焦或选择某个菜单项时，value 的值会被插入到<input>元素中。这种说法不仅可以从实验室页面中得到验证，而且可以从自动完成部件的源代码中看出端倪：

```
return matcher.test( value.label || value.value || value );
```

很明显，默认的匹配逻辑是将用户的输入值和 label 属性进行比对，而非作者一再强调的 value 属性。

② 原文错误，如①所述。

③ 原文错误，如①所述。

当我们想在菜单中使用短格式的值来呈现大量数据时，这种方式非常有用，但是对于很多情况（可能是大部分情况），标签和值将会是相同的。对于这些常见的情况，jQuery UI 允许指定数据为字符串数组，并将数组中的字符串值同时作为标签和值。

提示　当提供对象时，如果只指定了 label 或 value 中的一个，那么提供的这个值就会被自动用作 label 和 value。

为了使部件能正常工作，提供的数据项不必要按照特定的顺序排列（例如分类数据），并且显示在菜单中匹配的数据项会按照它们在数组中出现的顺序排列。

当使用本地数据时，匹配算法是：任何包含用户输入数据（被称为*术语*）的候选值都被认为是匹配的。如果这不是你想要的（假设你只希望匹配以术语*开始*的值），也不用担心。还有两种更加通用的方式来提供数据源，可以让我们能够完全控制匹配算法。

对于第一种方案，我们可以指定数据源为服务器资源 URL，其返回的响应包含匹配术语的数据值，这个术语是作为名为 term 的请求参数传递给服务器资源的。返回的数据应该是一个 JSON 响应，其求值结果是所支持的本地数据格式中的一种，通常是字符串的数组。

注意，source 的这种变体执行搜索并*只*返回匹配的元素（不会对返回的数据做进一步处理）。无论返回什么样的结果，它们都会被原原本本地显示在自动完成的菜单中。

当需要最大程度的灵活性时，可以使用另一种方案：提供一个回调函数作为 source 选项，在部件需要数据的时候会调用这个回调函数。在调用回调函数时向其传入以下两个参数。

- 拥有单个属性 term（包含匹配的术语）的对象。
- 将要调用的回调函数，向其传递匹配的结果以显示菜单。这组结果可以是接受本地数据的任意格式，通常是字符串数组。

这种回调机制提供了最大程度的灵活性，因为我们可以使用任何机制和算法将术语转变为匹配的元素集。使用 source 的这种变体的代码框架如下所示：

```
$('#control').autocomplete({
  source: function(request,response) {
    var term = request.term;
    var results;

    // 用来填充结果数组的算法

    response(results);
  }
});
```

和 source 的 URL 变体一样，结果应该只包含那些将要显示在自动完成菜单中的值。

在自动完成实验中练习使用 source 选项。在实验室中，有关不同的 source 选项有以下几点需要注意。

- “local string”选项提供了包含 79 个值的列表，所有的值都是以字母 F 开头。
- “local object”选项提供了一个短列表，其中包含用户名作为标签、完整的名称作为值的

11

对象。注意，匹配是发生在标签上而不是值上[①]。（提示：输入字母 b 看看。）

- 对于 URL 变体，后台资源只会匹配以术语开头的值。与提供本地值的情况（术语可以显示在字符串中的任何位置）相比，它采用了不同的算法。这种区别是有意而为之的，而且为了强调后台资源可以自由地采取任何所喜欢的匹配标准。
- 回调函数变体只是简单地返回全部以 F 开头的 79 个本地选项。复制一份实验室页面，修改该回调函数来测试任何你所喜欢的、用来筛选返回值的算法。

自动完成部件在工作的时候触发了各种事件。下面就来看看有哪些事件。

11.4.3　自动完成部件的事件

在自动完成部件运行的时候，触发了很多自定义事件，这些事件不仅用来通知我们发生了什么，而且可以让我们取消某些操作。

和其他的 jQuery UI 自定义事件一样，向事件处理器传入两个参数：事件实例和自定义对象。除了 `autocompletefocus`、`autocompletechange` 和 `autocompleteselect` 事件，这个自定义对象是空的。对于 `focus`、`change` 和 `select` 事件，这个对象包含一个 `item` 属性，这个属性又包含 `label` 和 `value` 属性，分别表示聚焦或选择的值的 `label` 和 `value` 属性。所有事件处理器的函数上下文（`this`）都被设置为`<input>`元素，参见表 11-8。

表11-8　jQuery UI自动完成部件的事件

事　　件	选　　项	描　　述
autocompletechange	change	当选择一个菜单项导致`<input>`元素的值发生改变时触发。这个事件总会在`autocompleteclose`事件之后被触发
autocompleteclose	close	每当自动完成菜单关闭时触发
autocompletefocus	focus	每当一个菜单项获取焦点时触发。除非取消（例如，返回`false`），否则聚焦的值会被设置到`<input>`元素中
autocompleteopen	open	在已经准备好数据并且准备打开菜单时触发
autocompletesearch	search	在`delay`和`minLength`条件都已经满足之后，由`source`指定的机制在激活之前触发。如果取消，搜索操作将被终止
autocompleteselect	select	从自动完成菜单选择某项值时触发。取消这个事件会阻止该值被设置到`<input>`元素中（但是不会阻止关闭菜单）

自动完成实验室使用所有的这些事件来更新控制台显示（每当事件被触发时）。

下面来看看如何为自动完成部件设置样式 。

11.4.4　自动完成部件的样式

和其他部件一样，自动完成部件通过将类名分配给组成自动完成部件的元素，来继承 jQuery

① 如果此时输入字母 o，则自动完成菜单中显示的是 honcho 和 comedian，对应的数据源是`{label:"honcho", value:"Pointy-haired Boss"}`和`{label:"comedian", value:"Charlie Chaplin"}`两个对象，可见匹配是在 `label` 上进行的。

UI CSS 主题的样式元素。

当把<input>元素转变为自动完成部件时，就会向其添加 ui-autocomplete-input 类名。

自动完成菜单会被创建为带有类名 ui-autocomplete 和 ui-menu 的无序列表元素(<ul>)。菜单中的值被创建为带有类名 ui-menu-item 的<li>元素。在这些列表项中，锚点元素会在鼠标悬停时获得 ui-state-hover 类名。

我们可以使用这些类名在自动完成元素上添加自己的样式风格。

例如，假设我们希望把自动完成菜单的透明度稍微调低，可以使用如下的样式规则：

```
.ui-autocomplete.ui-menu { opacity: 0.9; }
```

在此要小心。如果元素透明度过低会使其变得不可见①。

如果有很多匹配项的话，自动完成菜单会变得非常大。如果希望用更少的空间来显示更多的项目，那么可以使用如下规则缩小每一项的字体大小：

```
.ui-autocomplete.ui-menu .ui-menu-item { font-size: 0.75em; }
```

注意，ui-menu-item 不是特定于自动完成部件的类名（如果是的话，它会包含文本 autocomplete），因此我们使用 ui-autocomplete 和 ui-menu 来限定此类名，以确保不会在无意中将样式应用到页面上的其他元素上。

如果希望让鼠标悬停的项目突出显示，该怎么办？可以改变它们的表框为红色：

```
.ui-autocomplete.ui-menu a.ui-state-hover { border-color: red; }
```

自动完成部件可以让用户在大型数据集中快速地检索，从而防止了信息的泛滥。下面来看看如何简化另一件在数据输入方面一直都很麻烦的事情：输入日期。

11.5 日期选择器

输入日期信息是另一个一直令 Web 开发人员感到焦虑的领域，输入的友好度也往往令用户失望。有很多使用基本 HTML 4 控件的方法，但是所有的尝试都有自身的缺点。

很多站点为用户呈现了一个简单的文本输入框，用户必须向其输入日期。但是尽管包含了类似“请输入 dd/mm/yyyy 格式的日期”的说明，用户仍然还是会弄错。同样，一些 Web 开发人员自己也会弄错。如果在经过 15 次失败的尝试后，才发现在输入单个数字的天数或月份时必须包含前面的零，你是不是有一种想把自己的电脑从屋子中丢出去的冲动？

另一个方法是使用三个下拉列表，分别表示月份、天数和年份。尽管这种方法极大地减少了用户出错的可能性，但是却很笨拙且需要多次单击才能选择一个日期。而且开发者仍然需要防止出现类似“2 月 31 日”的选项。

当人们想起日期，就会想到日历，因此让用户输入日期最自然的方式应该是让他们在一个日历中选择。

① opacity 的取值在 0 ~ 1 之间，取值为 1 的元素完全不透明，取值为 0 是完全透明，不可见的。
——编者注

通常称为日历控件或日期选择器的脚本已经存在有一段时间了，但是它们通常不好配置，不适合在页面上使用，而且还需要和站点的样式匹配。把这个问题留给 jQuery 和 jQuery UI 吧，使用 jQuery UI 的日期选择器很容易就能搞定它。

11.5.1　创建jQuery日期选择器

创建 jQuery 日期选择器很简单，尤其是如果采用默认值的话。只有在为了更好地配合应用程序而配置日期选择器的众多选项时才会有些复杂。

和其他的 jQuery UI 元素一样，`datepicker()`公开了基本的 UI 方法集，同时也提供了一些特殊的方法用来在创建之后控制元素。

命令语法：datepicker

```
datepicker(options)
datepicker('disable')
datepicker('enable')
datepicker('destroy')
datepicker('option', optionName, value)
datepicker('dialog', dialogDate, onselect, options, position)
datepicker('isDisabled')
datepicker('hide', speed)
datepicker('show')
datepicker('getDate')
datepicker('setDate', date)
datepicker('widget')
```

将包装集中的<input>、<div>和<span>元素转变为日期选择控件。对于<input>元素，日期选择器会在输入框获取焦点时显示；对于其他元素，会创建一个内联的日期选择器

参数

options	（对象）一个散列对象，由要应用到包装集中元素的选项组成（参见表 11-9），从而使这些元素成为日期选择器
'disable'	（字符串）禁用日期选择控件
'enable'	（字符串）重新启用被禁用的日期选择控件
'destroy'	（字符串）使任何转变为日期选择控件的元素恢复到其初始状态
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是日期选择元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数

optionName	（字符串）要设置或返回的选项名称的值（参见表 11-9）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
value	（对象）指定要设置选项的值（通过 optionName 参数来标识）
'dialog'	（字符串）显示包含日期选择器的 jQuery UI 对话框
dialogDate	（字符串|日期）将对话框中日期选择器的初始日期指定为字符串表示的当前日期格式（查看表 11-9 关于 dateFormat 选项的描述）或 Date 实例
onselect	（函数）如果指定，则定义当选择日期时要调用的回调函数（向该函数传入日期文本和日期选择器实例）
position	（数组|事件）用来指定的对话框位置的数组（形式为[left, top]），或者用来确定对话框位置的鼠标事件的 Event 实例
'isDisabled'	（字符串）返回 true 或 false 来报告日期选择器当前是否被禁用
'hide'	（字符串）关闭日期选择器
speed	（字符串|数字）slow、normal、fast 或者一个毫秒值，用来控制关闭日期选择器的动画
'show'	（字符串）打开日期选择器
'getDate'	（字符串）返回日期选择器当前选中的日期。如果还没有选中任何值，则这个值可以是 null
'setDate'	（字符串|日期）将指定的日期设置为日期选择器的当前日期
date	（字符串|日期）为日期选择器设置的日期。这个值可以是 Date 实例，也可以是用于识别绝对或相对日期的字符串。绝对日期使用控件的日期格式（由 dateFormat 选项来指定，参见表 11-9）来指定，或者是由相对当前日期的值组成的字符串。相对日期由数字组成，后面紧跟着表示月份的 m、表示天数的 d、表示星期的 w 以及表示年份的 y 例如，明天是+1d、一周零 4 天是+1w +4d。正负值都可以使用
'widget'	（字符串）返回日期选择部件元素（即带有 ui-datepicker 类名的元素）

返回值

包装集，除了返回如上述值的情况

似乎是为了弥补自动完成部件选项过少的遗憾，日期选择器提供了一组令人眼花缭乱的选项，使其成为 jQuery UI 集合中可配置性最高的部件。不用太担心，通常只需使用默认值。不过在需要更好地适应站点时，可以修改这些选项，改变日期选择器的工作方式。

不过这些选项产生了一个相当复杂的日期选择实验室页面（参见图 11-11）。你可以在文件 chapter11/datepickers/lab.datepickers.html 找到这个页面。

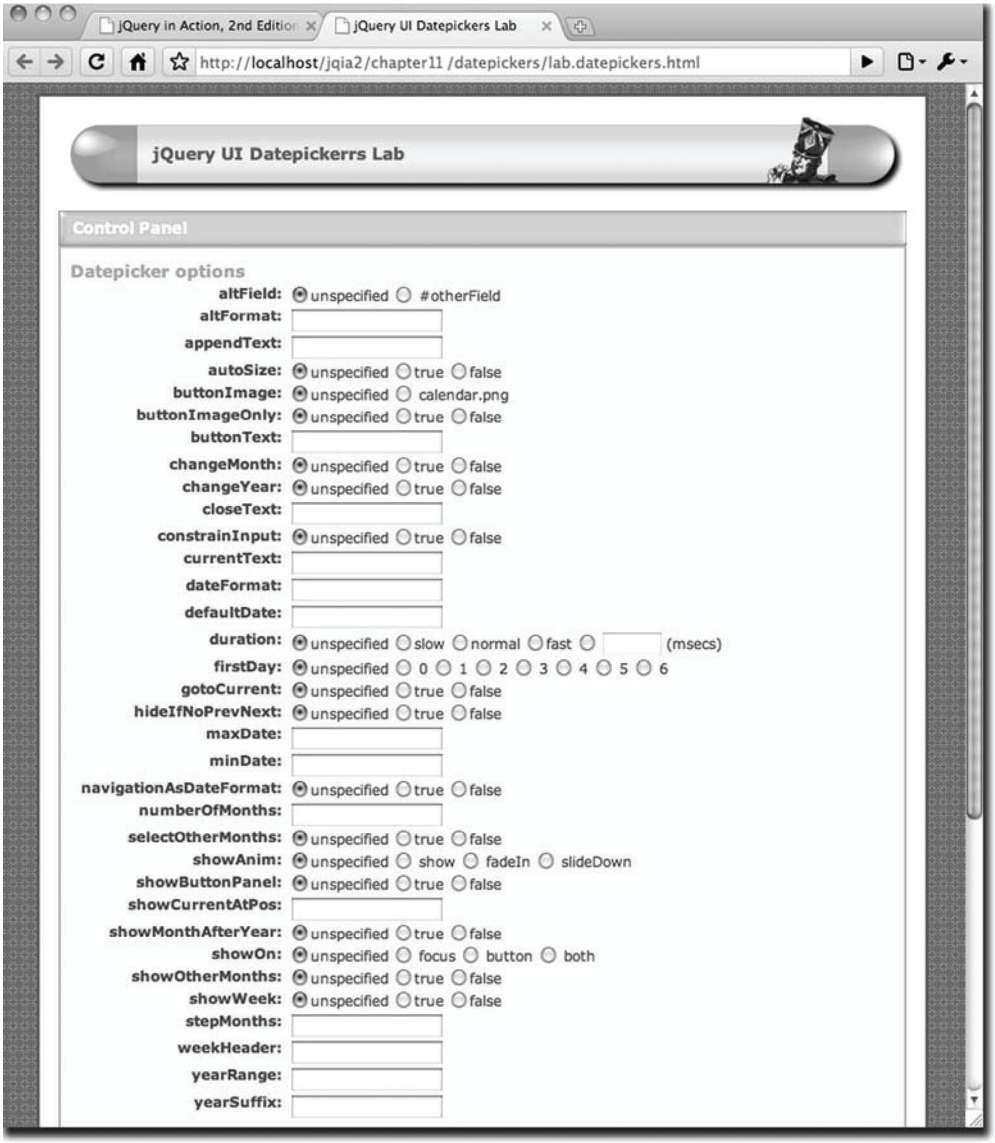


图 11-11　jQuery UI 日期选择实验室帮助我们理解日期选择控件各种可用的丰富选项（这个页面太长了，一个屏幕根本显示不下）

你可以一边阅读表 11-9 中所描述的那些选项，一边在日期选择实验室页面尝试运用这些选项。

表11-9 jQuery UI日期选择器的选项

选 项	描 述	是否在实验室页面中
altField	（选择器）为字段指定的jQuery选择器，这个字段的值会在选择日期时同时更新。使用altFormat选项来设置这个值的格式。这个选项对于将日期值设置到一个隐藏输入元素（将要提交到服务器）非常有用，同时向用户显示一个更加用户友好的格式	✓
altFormat	（字符串）当指定altField时，提供将要被写入另一个元素的日期格式。这个格式和$.datepicker.formatDate()实用函数使用的格式相同——详细信息请参见11.5.2节中的描述	✓
appendText	（字符串）被放置到<input>元素后面的值，用来向用户显示说明。这个值显示在带有类名ui-datepicker-append的<span>元素中，并且可以包含HTML标签	✓
autoSize	（布尔）如果为true，则调整<input>元素的尺寸来适合dateFormat选项指定的日期选择器的日期格式。如果省略，则不会改变<input>元素的尺寸	✓
beforeShow	（函数）在日期选择器显示之前被调用的回调函数，并将<input>元素和日期选择器实例作为参数传入回调函数。这个函数可以返回一个改变日期选择器的选项散列值。	✓
beforeShowDay	（函数）在日期选择器显示日期之前被调用的回调函数，并将日期作为唯一的参数传入回调函数。这可以用来覆盖显示某些日期元素的默认行为。这个函数必须返回一个包含三个元素的数组，如下所示： ❑ [0]——true使得该日期可选，false使其不可选 ❑ [1]——应用到日期元素的以空格分隔的CSS类名字符串，或者不应用CSS规则的空白字符串 ❑ [2]——可选的字符串为日期元素添加提示文本	
buttonImage	（字符串）指定显示在按钮上的图片路径，将showOn选项设置为button或both都可以启用这个功能。如果同时提供了buttonText，则按钮文本将成为按钮的alt特性	✓
buttonImageOnly	（布尔）如果为true，则只显示由buttonImage指定的图片（不是在按钮上显示图片）。要显示图片，showOn选项必须是button或both两者之一	✓
buttonText	（字符串）指定显示在按钮上文本内容，将showOn选项设置为button或both能够启用这个功能。如果同时提供了buttonImage，则此文本将变成图片的alt特性	✓
calculateWeek	（函数）自定义函数，用来计算并返回作为单个参数传入的日期所在的周数。默认是由$.datepicker.iso8601Week()实用函数提供的实现	
changeMonth	（布尔）如果为true，则显示月份的下拉列表，允许用户直接改变月份，而无需使用箭头按钮来逐个遍历月份。如果省略，则不显示月份下拉列表	✓
changeYear	（布尔）如果为true，则显示年份的下拉列表，允许用户直接改变年份而无需使用箭头按钮来逐个遍历年份。如果省略，则不显示年份下拉列表	✓
closeText	（字符串）如果通过showButtonPanel选项来显示按钮面板，则用指定文本，替换关闭按钮默认标题Done	✓
constrainInput	（布尔）如果为true（默认值），则限制输入到<input>元素的文本为控件所允许日期格式的字符（参考dateFormat选项）	✓

（续）

选　　项	描　　述	是否在实验室页面中
currentText	（字符串）如果通过showButtonPanel选项来显示按钮面板，则用指定文本替换当前按钮默认文本Today	✓
dateFormat	（字符串）指定所使用的日期格式。详细信息请参见11.5.2节	✓
dayNames	（数组）7个元素所组成的数组，提供一周内每天的名称，其中第一个元素表示星期天。可以用来本地化控件。默认集合是由英文表示的完整的每天名称组成的数组	
dayNamesMin	（数组）7个元素所组成的数组，提供一周内每天名称的缩写，其中第一个元素表示星期天，用作列的标题。可以用来本地化控件。默认集合是由英文表示的每天名称前面两个字母组成的数组	
dayNamesShort	（数组）7个元素所组成的数组，提供短格式一周内每天的名称，其中第一个元素表示星期天。可以用来本地化控件。默认集合是由英文表示的每天名称前面三个字母组成的数组	
defaultDate	（日期|数字|字符串）如果<input>元素没有值的话，设置控件的初始日期，用来覆盖默认值（当前日期）。它可以是date实例，距离今天的天数，或者指定绝对或相对日期的字符串。详细信息请参见datepicker()方法语法中date参数的描述	✓
disabled	（布尔）如果指定并设置为true，则部件初始是被禁用的	
duration	（字符串|数字）指定显示日期选择器的动画速度。可以是表示动画持续时间长短的slow、normal（默认值）、fast或毫秒数	✓
firstDay	（数值）设置一周中的第一天，并且会在最左侧的一列显示。星期天（默认值）是0，星期六是6	✓
gotoCurrent	（布尔）如果为true，则当天的链接被设置为选中的日期，覆盖其默认值（当前日期）[①]	✓
hideIfNoPrevNext	（布尔）如果为true，则在前一个月和后一个月的链接不适用时隐藏它们（而不是仅仅禁用这些链接），由minDate和maxDate选项的设置决定。默认为false	✓
isRTL	（布尔）如果为true，则本地化指定一个从右向左的语言。由这个控件的本地化版本所用。默认为false	
maxDate	（日期|数字|字符串）为控件设置最大可选择的日期。它可以是Date实例，距离今天的天数，或者指定绝对或相对日期的字符串。详细信息请参阅日期选择器setDate方法语法中date参数的描述	✓
minDate	（日期|数字|字符串）为控件设置最小可选择的日期。它可以是Date实例，距离今天的天数，或者指定绝对或相对日期的字符串。详细信息请参阅datepicker()方法语法中date参数的描述	✓
monthNames	（数组）12个元素所组成的数组，提供完整月份名称，其中第一个元素表示一月。可以用来本地化控件。默认集合是由英文表示的完整月份名称组成的数组	
monthNamesShort	（数组）12个元素所组成的数组，提供短格式月份名称的，其中第一个元素表示一月。可以用来本地化控件。默认集合是由英文表示的完整月份名称前三个字母组成的数组	✓

① 必须同时将 showButtonPanel 设置为 true，才能显示当天的链接。

（续）

选　项	描　述	是否在实验室页面中
navigationAsDateFormat	（布尔）如果为true，则导航链接nextText、preText和currentText会在显示之前传入$.datepicker.formatDate()函数进行格式化。这将可以为这些选项提供日期格式字符串，并将其转变为相应的日期值①。默认为false	✓
nextText	（字符串）指定文本来替换下个月导航链接的默认文本Next。注意，ThemeRoller会将这个文本替换为一个图标	✓
numberOfMonths	（数值|数组）在日期选择器中显示的月份个数，或者指定月份网格中行数和列数的二维数组。例如，[3,2]将会在3行2列的网格中显示6个月。默认情况下，只显示一个月	✓
onChangeMonth-Year	（函数）当日期选择器转到一个新的月份或年份时调用的回调函数，向此回调函数传递的参数可以是选择的年份、月份（从1开始算起），或日期选择器实例。函数上下文被设置为输入字段元素	
onClose	（函数）每当日期选择器关闭时调用的回调函数，向此回调函数传递的参数可以是选择的日期（以文本表示，如果没有选择，则为空字符串），或日期选择器实例。函数上下文被设置为输入字段元素	
onSelect	（函数）每当选择一个日期时调用的回调函数，向此回调传递的参数可以是选择的日期（以文本表示，如果没有选择，则为空字符串），以及日期选择器实例。函数上下文被设置为输入字段元素	
prevText	（字符串）指定文本来替换上个月导航链接默认文本Prev（注意，ThemeRoller会将这个文本替换为一个图标）	✓
selectOtherMonths	（布尔）如果为true，则显示在当前月之前和之后的日期是可选的。只有将showOtherMonths选项设置为true，才显示这些日期。默认情况下，这些日期是不可以选择的	✓
shortYearCutoff	（数值|字符串）如果是一个数字，则指定0～99之间的一个年份，大于这个值的两位数字的年份值都被认为是上一个世纪②。例如，如果指定为50，则年份39被认为是2039，而年份52被解释为1952。 如果是一个字符串，则此字符串值被转化为整数值后再与当前年份相加 默认值是+10，表示在当前年份的基础上再增加10年作为此选项的值	
showAnim	（字符串）设置显示、隐藏日期选择器的动画名称。如果指定，必须是show（默认值）、fadeIn、slideDown或任何jQuery UI显示/隐藏动画名称中的一个	✓

① 作为示例，请在实验室页面运行这段代码：

```
$('.testSubject').datepicker({navigationAsDateFormat:true,showButtonPanel:true,
currentText:"yy-mm-dd"});
```

你会看到原本显示“Today”的按钮现在显示的是当前的日期，比如“2011-11-18”。

② 任何小于或等于 shortYearCutoff 选项值的 2 位数字的年份都被认为是本世纪，而大于此选项值的 2 位数字的年份被认为是上个世纪。作为演示，请在实验室页面执行如下代码（你可能需要借助 Firefox 中的 Firebug 来完成这个操作）：

```
$(".testSubject").datepicker({shortYearCutoff:11,dateFormat:"dd-mm-yy",
defaultDate:"01-09-12"});
```

然后单击文本框打开日期选择器，你会发现日期选择器的标题显示为 September 1912（而不是 September 2012）。

11

（续）

选　　项	描　　述	是否在实验室页面中
showButtonPanel	（布尔）如果为true，则显示日期选择器底部的按钮面板，包含当前按钮和关闭按钮。这些按钮的标题由currentText和closeText选项设置。默认为false	✓
showCurrentAtPos	（数值）在多月份显示中，指定当前月份的显示位置，从左上角开始的从零算起的索引位置①。默认为0	✓
showMonthAfterYear	（布尔）如果为true，则日期选择器头部的月份和年份的位置互换。默认为false	✓
showOn	（字符串）指定什么事件触发日期选择器的显示，可以是focus、button或both其中之一 focus（默认值），在<input>元素获得焦点时显示日期选择器，而button会在<input>元素后创建一个按钮元素（在任何追加的文本之前），单击这个按钮元素会显示日期选择器。按钮的外观可以使用buttonText、buttonImage和buttonImageOnly选项改变 both不仅会创建触发器按钮，而且聚焦事件也会显示日期选择器	✓
showOptions	（对象）如果使用showAnim选项指定了一个jQuery UI动画，可通过此参数提供传入到这个动画的选项散列值	
showOtherMonths	（布尔）如果为true，则显示本月第一天和最后一天之前或之后的日期。这个日期是不可选择的，除非将selectOtherMonths选项设置为true。默认为false	✓
showWeek	（布尔）如果为true，则在月份的左侧显示一个包含星期数的列。calculateWeek选项可以用来修改生成这个值的方式。默认为false	✓
stepMonths	（数值）指定单击一个月份导航链接时，一次移动几个月。默认情况下，每次只移动一个月	✓
weekHeader	（字符串）当showWeek为true时，指定星期数列显示的文本，覆盖默认值Wk	✓
yearRange	（字符串）当changeYear为true时，指定显示在下拉列表中年份的数量（形式为from:to）。值可以是绝对或者相对的（例如，2005:+2指2005年到当前年份之后的2年）。可以使用前缀c来指相对于选中年份的值，而不是相对于当前年份的值（例如，c-2:c+3）	✓
yearSuffix	（字符串）指定显示在日期选择器头部年份后面的文本	✓

还没有对此感到厌烦吧？

尽管将这些选项作为一个整体来对待相当痛苦，但是大部分的日期选择器选项只会在需要覆盖默认值的情况下才会用到。在创建日期选择器的时候不指定任何选项的情况也是常见的。

11.5.2　日期选择器的日期格式

表 11-9 中的很多日期选择器选项使用字符串来表示*日期格式*。它们是用来格式化和解析日期的字符串。字符串中的字符模式代表了日期的各个部分（例如，y 代表年份，MM 代表完整的

① 在实验室页面中，通过如下步骤来观察这个选项的行为：设置 numberOfMonths 的值为 4，同时设置 showCurrentAtPos 的值为 2，单击 Apply。然后单击文本框弹出日期选择器，观察包含当前日期的月份位于从左侧开始的月份列表的第 3 个位置。

月份名称），或者是简单的模板文本（字面值）。

表 11-10 显示了用在日期格式模式中的字符模式以及它们所代表的含义。

表11-10　日期格式的字符模式

模　　式	描　　述
d	月份中的第几天，去除前面的零
dd	2个数字表示的月份中的第几天，值小于10时包含前面的零
o	一年中的第几天，去除前面的零
oo	3个数字表示的一年中的第几天，值小于100时包含前面的零
D	短格式的日期名称
DD	完整的日期名称
m	一年中的月份，去除前面的零，其中一月份是1
mm	2个数字表示的一年中的月份，值小于10时包含前面的零
M	短格式的月份名称
MM	完整的月份名称。
y	2个数字表示的年份，值小于10时包含前面的零
yy	4个数字表示的年份
@	自从1970-1-1以来的毫秒数
!	自从0001-1-1以来多少个100毫微秒
''	单引号
'...'	文本（单引号引用的字面值）
其他的任何字符	文本

日期选择器将一些大家都熟知的日期格式模式定义为常量，如表 11-11 所示。

我们会在 11.5.4 节中讨论日期选择器实用函数时重新探讨这些模式。

下面来关注一下日期选择器触发的事件。

表11-11　日期格式模式的常量

常　　量	模　　式
$.datepicker.ATOM	yy-mm-dd
$.datepicker.COOKIE	D,dd M YY
$.datepicker.ISO_8601	yy-mm-dd
$.datepicker.RFC_822	D, d M Y
$.datepicker.RFC_850	DD,dd-M-Y
$.datepicker.RFC_1036	D,d M Y
$.datepicker.RFC_1123	D, d M YY
$.datepicker.RFC_2822	D, d M YY
$.datepicker.RSS	D, d M Y
$.datepicker.TICKS	!
$.datepicker.TIMESTAMP	@
$.datepicker.W3C	yy-mm-dd

11.5.3　日期选择器的事件

太令人惊讶了，日期选择器没有事件！

jQuery UI 1.8 中的日期选择器代码是代码库中最古老的一部分代码了，它还没有更新过，并不守其他部件所遵守的现代事件触发的约定。预计这会在 jQuery UI 的未来版本中改变，根据 jQuery UI 路线图状态（可以在 http://wiki.jqueryui.com/中找到此路线图），这个部件会在 2.0 版本中彻底重写。

目前来说，在日期选择器发生感兴趣的事情时指定回调函数的选项有 `beforeShow`、`beforeShowDay`、`onChangeMonthYear`、`onClose` 和 `onSelect`。所有的这些回调函数都是通过这些选项来调用的（调用时将`<input>`元素作为函数上下文）。

尽管日期选择器缺少其他部件都支持的事件触发，但是它却给了我们额外的好处：一组有用的实用函数。下面来看看这些实用函数能做什么。

11.5.4　日期选择器的实用函数

日期是难以处理的数据类型。试想想一下处理闰年与非闰年的区别，不同天数的月份，不能被平均分配到每个月份的星期，以及所有其他折磨日期信息的古怪用法。幸运的是，JavaScript 的 `Date` 实现为我们处理了大部分细节问题。但是还有少数不尽如人意的地方——格式化和解析日期值就是其中的两个。

jQuery UI 日期选择器的出现填补了这些空白。从实用函数上来看，jQuery UI 不仅提供了格式化和解析日期值的方法，而且使拥有多个日期选择器的页面更容易处理大量的日期选择器选项。

下面就从如何处理大量选项入手。

1. 设置日期选择器的默认值

当需要使用多个选项来获取想要的外观和行为时，为页面上的每个日期选择器剪切和粘贴相同的选项集合似乎是完全错误的。我们可以将 `options` 对象存储在一个全局变量中，并且在创建每个日期选择器时引用这个变量，不过 jQuery UI 为此提供了一个更好的解决方法：注册一组默认选项来取代已经定义的默认值。这个实用函数的语法如下所示。

命令语法：`$.datepicker.setDefaults`

`$.datepicker.setDefaults(options)`

为所有随后创建的日期选择器设置选项的默认值

参数

`options`	（对象）选项的散列对象，作为所有日期选择器的默认值

返回值

无

回想一下日期选择器选项列表，其中的一些选项用来指定如何显示日期值的格式。日期选择器通常都会设置这些选项，而 jQuery UI 让我们可以在一个地方集中设置这些选项。

2. 格式化日期值

可以使用`$.datepicker.formatDate()`实用函数来格式化任何日期值，该函数的定义如下所示。

命令语法：`$.datepicker.formatDate`

`$.datepicker.formatDate(format, date, options)`

将传入的日期值格式化为由 `format` 模式和 `options` 指定的字符串

参数

`format`	（字符串）日期格式的模式字符串，参见表 11-10 和表 11-11
`date`	（日期）需要格式化的日期值
`options`	（对象）选项的散列对象，用来为日期和月份的名称提供可选的本地化值。可使用的选项有 `dayNames`、`dayNamesShort`、`monthNames` 和 `monthNamesShort`。有关这些选项的详细信息请参见表 11-9。如果省略，则使用默认的英文名称

返回值

格式化的日期字符串

这就淘汰了我们在第 7 章创建的日期格式化器。不过没关系，我们从那个练习中学到了很多内容，而且还可以在不使用 jQuery UI 的项目中使用该日期格式化器。

日期选择器还提供了其他哪些技巧呢？

3. 解析日期字符串

将文本字符串转变为日期值和将日期值格式化为文本字符串一样有用（甚至更有用）。jQuery UI 为此提供了`$.datepicker.parseDate()`函数，它的语法如下所示。

日期选择器还提供了另外一个实用函数。

4. 获取一年中的第几星期

jQuery UI 使用由 ISO 8601 标准定义的算法来作为 `calculateWeek` 选项的默认算法。如果需要在日期选择器控件之外的场合使用这个算法，则可以使用 jQuery UI 向我们公开的`$.datepicker.iso8601Week()`函数。

命令语法：`$.datepicker.parseDate`

`$.datepicker.parseDate(format, value, options)`

使用传入的 format 模式和 options 将文本值转变为日期值

参数

format	（字符串）日期格式的模式字符串，参见表 11-10 和表 11-11
value	（字符串）需要解析的文本值
options	（对象）选项的散列对象，用来为日期和月份的名称提供可选的本地化值，同时也指定如何去处理 2 个数字的年份值。可能的选项有 shortYearCutoff、dayNames、dayNamesShort、monthNames 和 monthNamesShort。有关这些选项的详细信息请参见表 11-9。如果省略，则使用默认的英文名称，并且将 shortYearCutoff 的默认值+10

返回值

解析后的日期值

命令语法：`$.datepicker.iso8601Week`

`$.datepicker.iso8601Week(date)`

给定一个日期值，计算由 ISO 8601 定义的星期数（即一年中的第几个星期）

参数

date	（日期）将要用来计算星期数的日期值

返回值

计算的星期数

ISO 8601 对星期数的定义是从星期一开始的星期，一年中的第一个星期是包含一月四日的那个星期（换句话说，也就是包含第一个星期四的那个星期）。

我们已经看到，jQuery UI 部件可以使用直观的方式从用户那里收集数据。下面将注意力转移到帮助我们组织内容的部件。如果此时你感到有点疲倦的话，那么现在正是时候坐下来休息一会儿，吃些点心，喝一杯包含咖啡因的饮料。

当你准备好之后，先来看看 Web 上最常用来组织内容的方式——选项卡。

11.6　选项卡

选项卡可能根本无需介绍。作为一种导航方式，它们在 Web 上已经无处不在，其使用频率仅次于超链接元素。模仿实际存在的卡片索引选项卡，GUI 选项卡将内容按逻辑分组，允许我们在相同级别的逻辑分组之间快速地切换。

在以前艰苦的日子里，在选项卡面板之间切换需要刷新整个页面，但是今天我们可以仅仅使用 CSS 根据不同的情况来显示和隐藏元素，甚至根据需要使用 Ajax 来获取隐藏的内容。

事实证明，为了得到正确的结果，“只使用 CSS”需要相当多的工作量，因此 jQuery UI 给予了我们一个现成的选项卡实现，当然是和下载的 UI 主题相匹配的。

11.6.1　创建选项卡的内容

目前学到的大部分部件都是将一个简单的元素（例如<button>、<div>、<input>）转变为目标部件。选项卡（就其本质而言）使用的是一个更加复杂的 HTML 结构。

包含三个选项卡的选项卡集应该遵循如下典型的结构模式：

```
<div id="tabset">
  <ul>
    <li><a href="#panel1">Tab One</a></li>
    <li><a href="#panel2">Tab Two</a></li>
    <li><a href="#panel3">Tab Three</a></li>
  </ul>
  <div id="panel1">
    ... content ...
  </div>
  <div id="panel2">
    ... content ...
  </div>
  <div id="panel3">
    ... content ...
  </div>
</div>
```

❶ 包含选项卡和选项卡面板

❷ 定义选项卡

❸ 提供面板

这个结构由包含整个选项卡集的<div>元素❶组成，这个元素又包含两部分：一个包含选项卡列表项（<li>）的无序列表（<ul>）❷，以及一组<div>元素（和选项卡的面板❸）。

用来呈现选项卡的每个列表项都包含一个锚点元素（<a>），这个元素不仅用来定义选项卡与其相应面板之间的关系，而且还是一个可获得焦点的元素。这些锚点元素的 `href` 特性指定了一个 HTML 锚点散列值，可以作为 jQuery 的 `id` 选择器，用来选择这个选项卡所关联的面板。

每个选项卡在第一次被选择时，它的内容也可以使用 Ajax 请求来从服务器获取。在这种情况下，锚点元素的 `href` 指定了当前活动内容的 URL，并且无需在选项卡集中包含面板。

如果要创建三个选项卡的选项卡集，其中所有内容都从服务器获取，那么 HTML 标记如下所示：

```
<div id="tabset">
  <ul>
    <li><a href="/url/for/panel1">Tab One</a></li>
    <li><a href="/url/for/panel2">Tab Two</a></li>
    <li><a href="/url/for/panel3">Tab Three</a></li>
  </ul>
</div>
```

在这种情况下，会自动创建三个用来保存动态内容的<div>元素。你可以在锚点上放置title特性来控制分配给这些面板元素的 id 值。title 值中间的空格被替换为下划线后，会作为相应面板的 id 值。

你可以使用这个 id 值预先创建面板元素，选项卡会正确地标识这个面板，但是如果你不这么做，选项卡会自动生成这个面板。例如，如果按如下方式重新编写第三个选项卡，

```
<li><a href="/url/for/panel3" title="a third panel">Tab Three</a></li>
```

面板的 id 值将会是 a_third_panel。如果这个面板已经存在，就会使用现有的面板；否则会创建一个新面板。

可以在一个选项卡集中自由混合 Ajax 或非 Ajax 选项卡。

一旦准备好所有的基础标签，就可以使用 tabs()方法来创建选项卡集了（应用到选项卡集外部的<div>元素上），它的语法如下所示。

命令语法：tabs

tabs(options)
tabs('disable', index)
tabs('enable', index)
tabs('destroy')
tabs('option', optionName, value)
tabs('add', association, label, index)
tabs('remove', index)
tabs('select', index)
tabs('load', index)
tabs('url', index, url)
tabs('length')
tabs('abort')
tabs('rotate', duration, cyclical)
tabs('widget')

将选项卡集标签（本节前面所述的）转变为一组 UI 选项卡控件

参数

options	（对象）要应用到选项卡集的散列对象的选项，如表 11-12 所述
'disable'	（字符串）禁用一个或者全部选项卡。如果提供一个从零开始的下标，则只禁用被标识的选项卡。否则，禁用整个选项卡集
'enable'	（字符串）重新启用被禁用的选项卡或者选项卡集。如果提供从零开始的下标，则启用被标识的选项卡。否则启用整个选项卡集

'destroy'	（字符串）使任何转变为选项卡控件的元素恢复到其初始状态
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是选项卡元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
optionName	（字符串）要设置或者返回的选项名称的值（参见表 11-12）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
index	（数值）用来标识将要被操作的选项卡从零开始的下标。用在 tabs()方法的如下变体中：disable、enable、remove、select、add、load 和 url
'add'	（字符串）向选项卡集添加一个新的选项卡。新的选项卡将会被插入到 index 参数指定的选项卡前面。如果没有提供 index，新选项卡会被放置在选项卡列表的尾部
association	（字符串）指定这个选项卡要关联的面板。它可以是将现有元素转变为面板的 id 选择符，也可以是用来创建 Ajax 选项卡的服务器端资源 URL
label	（字符串）分配给新选项卡的标签
'remove'	（字符串）从选项卡集中删除下标标识的选项卡
'select'	（字符串）使下标标识的选项卡成为被选择的选项卡
'load'	（字符串）强制重新加载下标标识的选项卡，忽略缓存
'url'	（字符串）改变下标标识的选项卡的关联 URL。如果该选项卡不是 Ajax 选项卡，则它将成为 Ajax 选项卡
url	（字符串）返回选项卡内容的服务器端资源 URL
'length'	（字符串）返回包装集中第一次匹配的选项卡集中的选项卡个数
'abort'	（字符串）终止任何正在进行的 Ajax 选项卡加载操作以及任何正在运行的动画
'rotate'	（字符串）以指定的间隔设置选项卡自动循环
duration	（数值）选项卡集循环的间隔（以毫秒为单位）。传入 0 或 null 来停止一个正在进行的循环过程
cycle	（布尔）如果为 true，则循环会在用户选择某个选项卡后继续进行。默认为 false
'widget'	（字符串）返回选项卡部件元素，即带有 ui-tabs 类名的元素

返回值

包装集，除了返回如上所述的值的情况

和预想的一样，如此复杂的部件有相当多的选项（参见表 11-12）。

像往常一样，我们提供了一个选项卡实验室来帮助你学习整理 `tabs()`方法的选项。可以从文件 chapter11/tabs/lab.tabs.html 中找到这个实验室页面，显示如图 11-12 所示。



图 11-12　jQuery UI 选项卡实验室页面向我们展示了如何使用选项卡将一系列显示内容组织在多个面板中

注意 因为这个实验室使用了服务器 Ajax 操作，所以它必须运行在第 8 章为示例所创建的 Tomcat 实例下(注意 URL 中的端口号为 8080)。你也可以远程访问这个实验室页面：http://www.bibeault.org/jqia2/chapter11/tabs/lab.tabs.html。

可用于 `tabs()`方法的选项显示在表 11-12 中。

表11-12 jQuery UI选项卡的选项

选　项	描　述	是否在实验室页面中
add	（函数）指定在选项卡集上创建的函数，用来作为tabsadd事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
ajaxOptions	（对象）指定传入$.ajax()中任何额外的选项散列值（在选项卡集的任何Ajax加载操作进行过程中）。关于这些选项的细节，请参考第8章对$.ajax()方法的描述	
cache	（布尔）如果为true，任何通过Ajax加载的内容将会被缓存。否则，会重新加载Ajax的内容。默认为false	✓
collapsible	（布尔）如果为true，则选择已经处于选中状态的选项卡会导致它变为未选中状态，从而导致没有选项卡处于选中状态，还会折叠面板区。默认情况下，单击已经选中的选项卡没有任何作用	✓
cookie	（对象）如果提供，使用cookie插件来记住最后选中的选项卡，并且在页面重新加载时恢复选中的选项卡 这个对象的属性由cookie插件指定：name、expires（单位是天）、path、domain和secure 使用此选项时需要预先加载cookie插件（http://plugins.jquery.com/project/cookie）	✓
disable	（函数）指定在选项卡集上创建的函数，用来作为tabsdisable事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
disabled	（数组）包含从零开始的选项卡下标值，用来指定初始被禁用的选项卡。如果未指定selected选项（默认为0），则即使数组中包含下标0也不会禁用第一个选项卡，因为这个选项卡是默认选中的	✓
enable	（函数）指定在选项卡集上创建的函数，用来作为tabsenable事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
event	（字符串）指定用来切换选项卡的事件。通常是click（默认值）或mouseover，但是也可以指定其他事件，如mouseout事件（尽管有点奇怪）	✓
fx	（对象）指定一个散列对象，用来配合为选项卡添加动画效果的animate()方法。可以使用duration属性来指定动画方法的间隔值：毫秒数、normal（默认值）、slow或fast。也可以指定位于0.0～1.0数字区间内的opacity属性	
idPrefix	（字符串）如果选项卡锚点上没有title特性，则为动态内容的选项卡面板生成唯一的id值时指定前缀值。如果省略，则默认使用ui-tabs-	

（续）

选　　项	描　　述	是否在实验室页面中
load	（函数）指定在选项卡集上创建的函数，用来作为tabsload事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
panelTemplate	（字符串）动态创建选项卡面板时使用的HTML模板。这个行为可能是add方法的结果，也可能是自动创建Ajax选项卡的结果。默认情况下，使用模板" "	
remove	（函数）指定在选项卡集上创建的函数，用来作为tabsremove事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
select	（函数）指定在选项卡集上创建的函数，用来作为tabsselect事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
selected	（数字）初始选中的从零开始的选项卡下标。如果省略，则选中第一个选项卡。可以使用值–1在初始时不选中任何选项卡	✓
show	（函数）指定在选项卡集上创建的函数，用来作为tabsshow事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-13	✓
spinner	（字符串）当获取远程内容时显示在Ajax选项卡上的HTML字符串。默认字符串是"Loading…"（嵌入的表示省略号的HTML实体字符是Unicode字符。） 为了显示正在加载的提示，选项卡锚点元素的内容必须是元素。例如： Slow	✓
tabTemplate	（字符串）当使用add方法创建新的选项卡时用到的HTML模板。如果省略，则默认值为" #{label} " 在模板中，符号#{href}和#{label}会被传入add方法的值所代替	

你现在应该对本书中的各种实验室页面非常熟悉了，在选项卡实验室中尝试基本选项时应该不需要任何帮助了。不过我想确保你理解关于 Ajax 选项卡的一些重要的细微差别，因此下面列出了一些实验室练习题，你可以在尝试了基本选项后进行练习。

- **练习 1**——打开实验室页面，保持所有控件处于它们的默认状态，单击 Apply。Food 和 Slow 选项卡是 Ajax 选项卡，它们的面板在选中选项卡之后加载。

  单击 Food 选项卡。这个选项卡会从 HTML 源加载并立即显示出来。注意控制台的 tabsload 事件。它意味着已经从服务器加载了内容。

  单击 Flowers 选项卡，然后再次单击 Food 选项卡。注意，当内容再次从服务器加载时另一个 tabsload 事件是如何被触发的。
- **练习 2**——重置实验室。为 cache 选择 true 选项，单击 Apply。

  重复练习 1 的动作，注意这次 Food 选项卡只有在它第一次被选择时才加载。
- **练习 3**——重置实验室，保持所有控件的默认状态，单击 Apply。

  重复练习 1，只不过这次单击 Slow 选项卡而不是 Flowers 选项卡。Slow 选项卡从服务器资源加载，这需要 10 秒钟的时间。注意在长时间的加载操作中，默认值“Loading…”是如何显示的，以及 tabsload 事件是如何在接收到内容后触发的。

- **练习 4**——重置实验室，为 spinner 选项选择 image 值，单击 Apply。
  重复练习 3 的动作。这次的加载过程会在选项卡上显示一个`<img>`元素的 HTML。你一定不能错过这个效果。

11.6.2　选项卡事件

当用户单击选项卡时，我们可能想要得到通知，这样的理由举不胜举。例如，我们可能想要等到用户真正选择一个选项卡时才在选项卡内容上执行一些初始化事件。毕竟，为什么要在用户可能根本不会查看的选项卡上做大量内容初始化的工作呢？对于已经加载的内容道理相同。我们可能想要在内容加载后执行一些任务。

为了帮助我们在适当的时候操作选项卡和选项卡内容，表 11-13 列出了在选项卡集的生命周期内不同时间所触发的事件。向每个事件处理器传入的第一个参数是事件实例，第二个参数是自定义对象，其属性由三个元素构成：

- `index`——与事件关联的从零开始的选项卡索引；
- `tab`——与事件关联的选项卡锚点元素的引用；
- `panel`——与事件关联的选项卡面板元素的引用。

表11-13　jQuery UI选项卡事件

事　件	选　项	描　述
`tabsadd`	`add`	当向选项卡集添加新的选项卡时触发
`tabsdisable`	`disable`	当禁用选项卡时触发
`tabsenable`	`enable`	当启用选项卡时触发
`tabsload`	`load`	当一个Ajax选项卡内容加载后触发（甚至出错时也触发）
`tabsremove`	`remove`	当删除选项卡时触发
`tabsselect`	`select`	当单击某个选项卡被单击、将要成为选中的选项卡（除非这个回调函数返回`false`，这种情况下会取消本次选择）时触发
`tabsshow`	`show`	当显示选项卡面板时触发

作为示例，假设想要为通过 Ajax 加载的所有选项卡面板中的图片元素添加一个类名，可以使用建立在选项卡集上的单个 `tabsload` 处理器来实现：

```
$('#theTabset').bind('tabsload',function(event,info){
  $('img',info.panel).addClass('imageInATab');
});
```

这个小示例中有几个重要的知识点：

- `info.panel` 属性引用受到影响的面板；
- 触发 `tabsload` 事件时面板的内容已经加载完毕。

下面就来向这些元素添加一些 CSS 类名，以便我们使用这些类名来修改元素的样式。

11.6.3　修改选项卡样式

当创建一个选项卡集时，下面的 CSS 类名就被应用到各种组成元素上。

- ui-tabs——添加到选项卡集元素的类名。
- ui-tabs-nav——添加到包含选项卡的无序列表元素上的类名。
- ui-tabs-selected——添加到选中选项卡的列表项的类名。
- ui-tabs-panel——添加到选项卡面板的类名。

你是否认为默认渲染的选项卡尺寸太大了？可以使用如下样式规则将它们缩小到合适的尺寸：

```
ul.ui-tabs-nav { font-size: 0.5em; }
```

你想让选中的选项卡突出显示吗？尝试下面样式：

```
li.ui-tabs-selected a { background-color: crimson; }
```

选项卡是一个很棒的常用部件，用来组织包含相关内容的各种面板，以便用户每次只能看到一个面板。但是如果觉得这个部件太俗气，而想使用不太通用的外观和感觉来达到相同的目的，该怎么办？

下面介绍的手风琴部件可能正好满足你的需求。

11.7　手风琴部件

尽管手风琴这个词可能让你想起留着小胡子的男人在深情地演奏着小夜曲，但在这里它是一个部件的名称，用来每次呈现一个内容面板（就像选项卡一样），这样的布局令人联想到真实的手风琴乐器。

与在面板区域的上部显示一组选项卡不同，手风琴将可选项作为一系列堆放在一起的水平条来呈现，每个水平条的内容显示在当前水平条和下一个水平条之间。如果你已经使用过代码示例的索引页（根目录的 index.html），那你就已经看到过手风琴部件了，如图 11-13 所示。

和选项卡集一样，每次只能打开手风琴部件的一个面板。默认情况下，手风琴部件也会调整每个面板的尺寸以便部件所占用的空间都是一样的（在无论打开哪个面板的情况下）。这使得手风琴部件成为页面上比较受欢迎的部件之一。

下面来看看如何创建手风琴部件。



图 11-13 使用手风琴部件来组织本书中众多代码示例的链接

11.7.1 创建手风琴部件

和选项卡集一样，手风琴部件需要一个特别的 HTML 结构来创建。由于手风琴部件拥有一个不同的布局，而且为了确保在缺少 JavaScript 时页面能够渐进消退[①]（gracefully degrade），因此手风琴部件的源结构和选项卡集的源结构大不相同。

手风琴部件需要一个外部的容器（`accordion()`方法应用到的元素），其中包含成对出现的标题和相关的内容。它不是使用 `href` 值来将内容面板和标题关联起来，而是（默认）在每个标题后面紧跟着作为下一个兄弟节点的内容面板。

典型的手风琴部件的结构如下所示：

① 渐进消退（Gracefully Degrade）和不唐突的 JavaScript（Unobtrusive JavaScript）都是 Web 2.0 中的流行术语，详情请参阅维基百科：http://en.wikipedia.org/wiki/Graceful_degradation。

```
<div id="accordion">

  <h2><a href="#">Header 1</a></h2>
  <div id="contentPanel_1">  ... content ... </div>

  <h2><a href="#">Header 2</a></h2>
  <div id="contentPanel_2">  ... content ... </div>

  <h2><a href="#">Header 3</a></h2>
  <div id="contentPanel_3">  ... content ... </div>

</div>
```

注意，标题文本还是被嵌入在一个锚点元素中（为了给用户一个可获取焦点的元素），但是通常将锚点的 href 特性设置为#，而不用来关联标题和它的内容面板。（有一个选项使得锚点的 href 值非常有意义[①]，但是通常将其设置为#）。

accordion()方法的语法如下所示。

命令语法：accordion

accordion(options)

accordion('disable')

accordion('enable')

accordion('destroy')

accordion('option', optionName, value)

accordion('activate', index)

accordion('widget')

accordion('resize')

将手风琴 HTML 源结构（本节前面所述）转变为手风琴部件

参数

options	（对象）要应用到手风琴部件的散列对象的选项，参见表 11-14
'disable'	（字符串）禁用手风琴部件
'enable'	（字符串）重新启用一个被禁用的手风琴部件
'destroy'	（字符串）使任何转变为手风琴部件的元素恢复到其初始状态
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是手风琴元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
optionName	（字符串）要设置或返回的选项名称的值（参见表 11-14）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
'activate'	（字符串）激活（打开）由 index 参数标识的内容面板

① 这里指的应该是 navigation 选项，下文会有详细描述。

index	（数值|选择器|布尔）用来标识将要被激活的手风琴面板从零开始的下标，或标识面板的选择器，或 false（如果指定 collapsible 选项为 true 的话，会关闭所有的面板）
'widget'	（字符串）返回手风琴部件元素（也即是带有 ui-accordion 类名的元素）
'resize'	（字符串）重新计算部件的尺寸。应该在可能触发部件尺寸改变的事件发生时调用这个方法；例如，改变容器的尺寸

返回值

包装集，除了返回如上所述的值的情况

简短但功能强大的 accordion()方法的选项列表参见表 11-14。

在你通读表 11-14 中的选项列表时，可以在实验室中进行相应的练习。

表11-14　jQuery UI手风琴部件的选项

选　项	描　述	是否在实验室页面中
active	（数值|布尔|选择器|元素|jQuery）指定哪个面板是初始打开的。它可以是从零开始的面板下标，或者一种标识面板标题元素的方法：元素引用、选择器或jQuery包装集 如果指定为false，则在初始时不会打开任何面板（除非设置collapsible为false）	✓
animated	（字符串|布尔）当打开和关闭手风琴面板时使用的动画名称。可以是slide（默认值）、bounceslide或任何安装的缓动特效（如果已经包含在页面上）。 如果指定为false，则不使用动画	✓
autoHeight	（布尔）除非指定为false，否则所有面板的高度都被强制调整为最高面板的高度，所有的面板具有相同的尺寸。否则，各个面板保持它们的自然尺寸。默认为true	✓
clearStyle	（布尔）如果为true，则会在动画结束后清除height和overflow样式。要应用这个选项，必须设置autoHeight选项为false	
change	（函数）指定在手风琴部件上创建的函数，用来作为accordionchange事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-15	✓
changestart	（函数）指定在手风琴部件上创建的函数，用来作为accordionchangestart事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-15	✓
collapsible	（布尔）如果为true，单击已经打开的手风琴面板标题栏会关闭此面板，使得没有面板处于打开状态。默认情况下，单击已经打开的面板标题栏没有任何作用	✓
disabled	（布尔）如果指定并且为true，则手风琴部件初始是被禁用的	
event	（字符串）指定切换手风琴标题栏的事件。通常是click（默认值）或mouseover，但是也可以指定其他事件，如mouseout的事件（尽管有点奇怪）	✓
fillSpace	（布尔）如果为true，则手风琴会调整尺寸，完全填充父元素的高度，覆盖autoHeight选项的值	

（续）

选　　项	描　　述	是否在实验室页面中
header	（选择器|jQuery）指定选择器或元素来覆盖默认识别标题元素的模式。默认值是`">li>:firstchild,>:not(li):even"`。只有当你为手风琴指定一个不符合默认模式的源结构时才使用这个选项	
icons	（对象）用来定义显示在打开和关闭面板标题栏文字左侧的图标的对象。用来指定关闭的面板图标的属性名为`header`，而用来指定打开的面板图标的属性名为`headerSelected` 这些属性的值是用来标识图标类名的字符串，定义在按钮部件中参见11.1.3节。 默认情况下，`header`属性值是`ui-icon-triangle-1-e`，`headerSelected`的属性值是`ui-icon-triangle-1-s`	✓
navigation	（布尔）如果为`true`，则当前地址（`location.href`）被用来尝试匹配手风琴标题栏中锚点标签的`href`值。这可以用来在页面显示的时候打开特定的手风琴面板 例如，设置`href`的值为锚点散列值例如`#chapter1`（诸如此类），如果URL（或者收藏夹地址）的后缀拥有相同的散列值，就会在页面显示的时候打开相应的面板。代码示例的`index.html`页面使用了这种技术。试试看！将`index.html#chapter3`作为URL的一部分来访问示例页面	
navigation-Filter	（函数）如果`navigation`选项为`true`，覆盖默认的导航过滤器[1]。你可以使用这个函数将`navigation`选项所描述的行为改变成任何你所期望的行为 调用这个回调函数时没有任何参数，标题栏中的锚点标签被设置为函数上下文。返回`true`意味着一次导航匹配发生了	

我们提供了手风琴实验室页面（在文件 chapter11/accordions/lab/accordions.html 中），用来演示很多选项的用法。页面如图 11-14 所示。

在学习一些基本的选项以及在手风琴实验室页面尝试运用这些选项之后，下面列出了一些练习题，你千万不能错过。

- **练习 1**——加载实验室，保持所有设置的默认值，单击 Apply。随意选中各个标题栏，注意当面板打开和关闭时，手风琴部件本身是如何保持自身尺寸不变化的。
- **练习 2**——重置实验室，为 `autoHeight` 选择 `false`，单击 Apply。进行练习 1 中的动作，注意这次当 Flowers 面板打开的时候，手风琴部件的高度缩小，正好适合 Flowers 面板内容的尺寸（比较短）。

接下来该处理那些在操作手风琴部件时所触发的事件了。

① 查看 jQuery UI 的源代码，你会发现这个默认函数非常简单：

```
navigationFilter: function() {
    return this.href.toLowerCase() === location.href.toLowerCase();
}
```

同时也请注意函数上下文（`this`）被设置成了标题栏中的锚点元素。

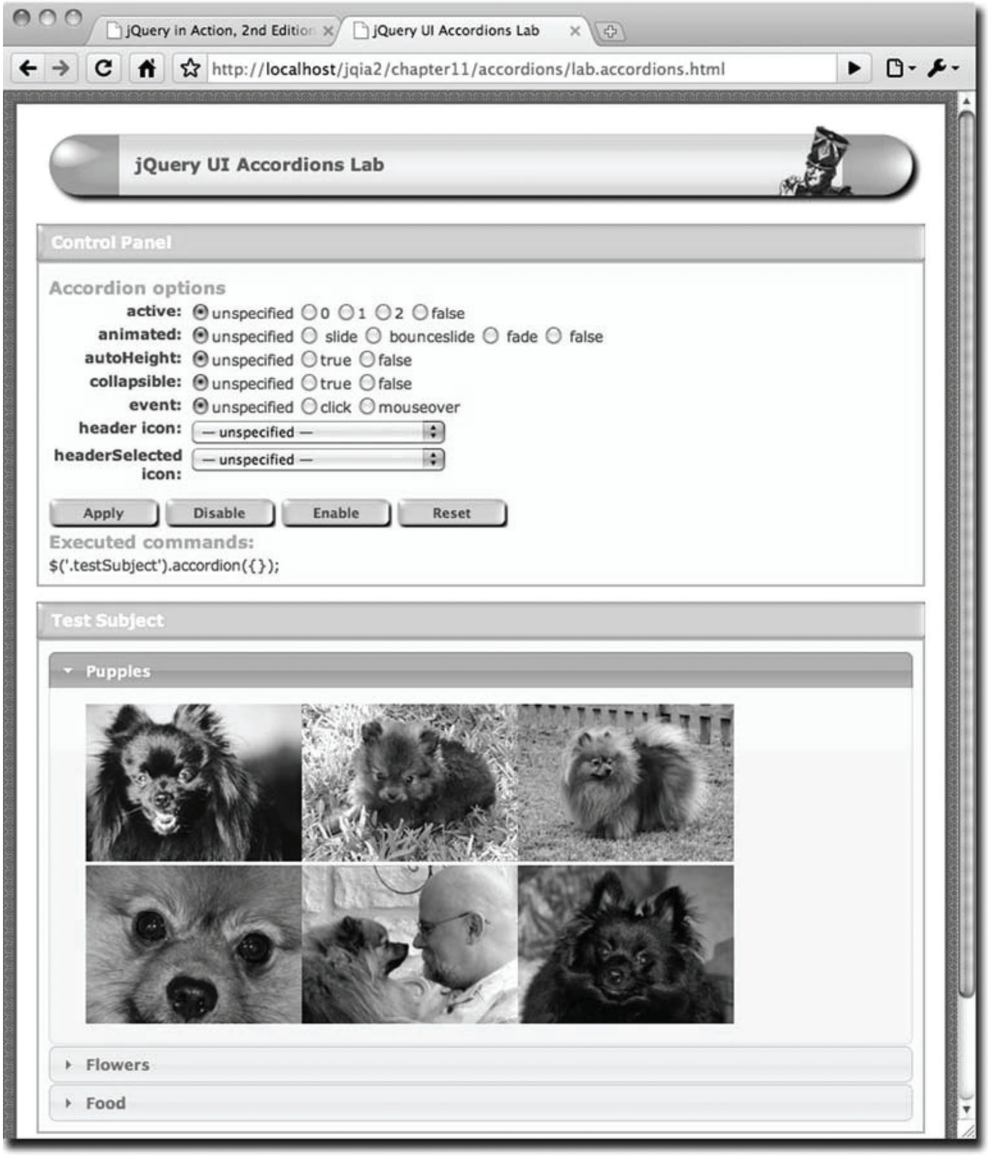


图 11-14　jQuery UI 手风琴实验室说明了如何以新颖时尚的方式向用户展示一系列的内容面板

11.7.2　手风琴部件的事件

当用户打开和关闭面板时，手风琴部件只会触发两个类型的事件，参见表 11-15。

向每个处理器传入通常的事件实例和自定义对象。对于两个事件类型而言，自定义对象都由下面这些属性组成。

- options——创建部件时传入 accordion()方法的选项。
- oldHeader——包含之前打开面板的标题元素的 jQuery 包装集。如果之前没有打开任何面板，则这个属性为空。
- newHeader——包含当前打开面板的标题元素的 jQuery 包装集。对于可折叠的手风琴部件[①]，当所有面板都关闭时，这个属性为空。
- oldContent——包含之前打开面板引用的 jQuery 包装集。
- newContent——包含当前打开面板引用的 jQuery 包装集。

表 11-15 列出了手风琴部件的各种事件。

表11-15　jQuery UI手风琴部件的事件

事　　件	选　　项	描　　述
accordionchangestart	changestart	当手风琴的选择项将要改变时触发
accordionchange	change	在手风琴的选择项已经改变后（在用来改变显示的任何动画过程结束之后）触发

这是一个相当短的事件列表，它也的确提出了一些挑战。例如，当打开初始面板（如果存在的话）时，我们不会收到任何通知，这一点很让人失望。我们将会看到当试图使用这些事件来操作手风琴部件时，事件变得有点麻烦。但是在学习这些事件来为手风琴部件添加功能的示例之前，先来考察一下 jQuery UI 向构成手风琴部件的元素所添加的 CSS 类名。

11.7.3　手风琴部件的样式类名

和选项卡部件一样，jQuery UI 向组成手风琴部件的元素添加了很多 CSS 类名。这些类名不仅可以用来为手风琴添加样式，而且可以用来通过 jQuery 选择器查找元素。在上一节学习如何找出涉及手风琴事件的面板时，我们就曾见到过类似的示例。

下面是应用到手风琴元素的类名。

- ui-accordion——添加到手风琴部件外部容器（在其上调用 accordion()方法的元素）的类名。
- ui-accordion-header——添加到成为可单击的所有标题元素的类名。
- ui-accordion-content——添加到所有面板元素的类名。
- ui-accordion-content-active——分配给当前打开面板元素（如果存在的话）的类名。
- ui-state-active——分配给当前打开面板（如果存在的话）的标题元素的类名。注意这是多个部件共享的通用 jQuery UI 类名中的一个。

使用这些类名，可以重新将手风琴元素改造成我们喜欢的样子，就像对选项卡所做的那样。请亲自尝试改变元素的样式：例如标题文字，或者围绕面板的边框。

① 可折叠的手风琴部件，也即是在调用 accordion()方法时设置 collapsible 选项为 true。

下面来看一下这些类名是如何帮助我们为手风琴部件添加功能的。

11.7.4　使用Ajax加载手风琴面板

手风琴部件缺少的一个特征（存在于它的“兄弟”选项卡部件中），是源生通过 Ajax 加载内容的能力。为了不让手风琴部件“忍受”这种自卑感，下面就来看看如何使用现有的知识轻松地为手风琴部件添加这个功能。

选项卡部件使用锚点标签的 `href` 来指定远程内容的地址。然而，手风琴部件忽略了标题栏中锚点标签的 `href` 值，除非使用 `navigation` 的情况。理解这一点，我们将可以安全地使用它来指定需要加载到面板中任何远程内容的地址。

这是一个不错的决定，因为它和选项卡的工作方式保持一致（一致性是件好事情），并且它意味着无需引入自定义选项或特性来记录这个地址。我们将保持“普通”面板的 `href` 特性值为`#`。

我们想要在每当将要打开面板时加载面板的内容，因此使用如下代码向手风琴部件（可能有多个）绑定 `accordionchangestart` 事件：

```
$('.ui-accordion').bind('accordionchangestart',function(event,info){
  if (info.newContent.length == 0) return;
  var href = $('a',info.newHeader).attr('href');
  if (href.charAt(0) != '#') {
    info.newContent.load(href);
    $('a',info.newHeader).attr('href','#');
  }
});
```

在这个处理器中，首先通过 `info.newContent` 提供的引用找到打开的面板。如果不存在打开的面板（可能存在于可折叠的手风琴部件中），则返回。

然后通过 `info.newHeader` 提供的引用上下文找到<a>元素（也即是找到激活标题栏中的锚点元素），并获取它的 `href` 特性。如果此特性值不是以`#`开头的，我们就假设它是一个提供面板内容的远程 URL。

为了加载远程内容，我们使用 `load()`方法，然后改变锚点的 `href` 特性值为`#`。最后这个动作是为了防止下次打开面板时再次从远程获取内容。（为了强制每次重新加载内容，需要删除 `href` 赋值语句。）

当使用这个处理器时，为了防止出现问题，在预先不清楚最大面板尺寸的时候，我们可能想要关闭 `autoHeight` 选项。这种实现的可运行示例位于文件 chapter11/accordions/ajax/ajax-accordion.html 中。

像往常一样，总会有不同方式的实现。尝试下面这个练习。

- **练习 1**——如果你想避免使用 `href` 值，以便可以使用 `navigation` 选项，要如何重新编写这个示例来使用自定义特性（或者你选定的其他策略）？

手风琴部件向我们提供了除选项卡面板之外的另一种选择，用来连续地向用户呈现相关的内容。现在让我们结束对这个部件的讨论，来看看另一个动态呈现内容的部件。

11.8　对话框

对话框这个概念根本无需介绍。自从图形用户界面（GUI）出现以来，对话框（无论是模式的还是非模式的）就是桌面应用程序设计的一个常用的主要手段，用来从用户获取信息或向用户传递信息。

然而，在 Web 界面中除了内置的 JavaScript 警告提示和确认工具，对话框并不是本来就有的概念。这些工具由于各种原因被认为是不合适的，其中主要的原因是不能修改它们的样式来和站点的样式保持一致，因此这些工具一般只用作调试助手。

IE 浏览器引入了基于 Web 对话框的概念，但是它不足以吸引标准社区的注意，仍然只是一个私有的解决方案[①]。

多年来，Web 开发者使用 `window.open()`方法来创建代表对话框的新窗体。虽然充满了问题，但是这种方法作为非模式对话框的解决方案是合适的，然而却无法像真正的模式对话框那样工作。

随着 JavaScript、浏览器和 DOM 操作的流行，开发者本身也变得强大起来，能够使用这些基本的工具来创建“浮动”在页面上的元素（甚至是以模式的方式来锁定输入元素），这更加接近非模式和模式对话框的语义。

因此，虽然在概念上，对话框在 Web 界面中并不是真实存在的，但是我们通过自己的努力使对话框看起就像存在于 Web 界面中。

下面来看看在这个领域 jQuery UI 为我们提供了什么。

11.8.1　创建对话框

尽管页面内对话框的想法看起来很简单[只是从页面流中删除一些内容，使用一个高的 z-index 将其浮动起来，然后为其添加一个“边框”（chrome）]，但是还有很多细节需要考虑。幸运的是，jQuery UI 会处理这些细节，允许我们创建非模式和模式对话框，并带有高级的特征，例如轻松地改变尺寸和重新定位的的能力。

注意　当应用对话框术语“边框”的时候，它代表的是包含对话框和允许操作对话框的框架和部件。这包括的特征如可改变尺寸边框、标题栏以及无处不在的用于关闭对话框的小“x”图标。

与选项卡部件和手风琴部件不同，对话框并不严格限制应用的元素几乎任何元素都可以成为它的主体，但通常将包含内容的`<div>`元素作为对话框的主体。

为了创建对话框，首先把将要成为对话框主体的内容选择到一个包装集中，然后对此包装集应用 `dialog()`方法。`dialog()`方法的语法如下所示。

① 这里指的应该是 IE 下 `window.showModalDialog()`和 `window.showModelessDialog()`两个方法。

命令语法：**dialog**

dialog(options)

dialog('disable')

dialog('enable')

dialog('destroy')

dialog('option', optionName, value)

dialog('open')

dialog('close')

dialog('isOpen')

dialog('moveToTop')

dialog('widget')

通过将包装集中的元素从文档流中删除，并将它们包含在“边框”中，从而将这些元素转变为对话框。注意,创建对话框的同时会自动打开这个对话框,除非将autoOpen选项设置为false来禁用自动打开功能

参数

options	（对象）一个散列对象，由要应用到对话框的选项组成（参见表 11-16）
'disable'	（字符串）禁用对话框
'enable'	（字符串）重新启用被禁用的对话框
'destroy'	（字符串）销毁对话框。一旦对话框被销毁，就不能再次打开它了。注意，销毁对话框不会使包含的元素恢复到正常的文档流中
'option'	（字符串）基于剩余的参数，设置包装集中所有元素的选项值，或者获取包装集中第一个元素（必须是对话框元素）的选项值。如果指定这个字符串作为第一个参数，则至少需要提供 optionName 参数
optionName	（字符串）要设置或返回的选项名称的值（参见表 11-16）。如果提供 value 参数，则这个值就成为选项的值。如果没有提供 value 参数，则返回已命名选项的值
'open'	（字符串）打开一个关闭的对话框
'close'	（字符串）关闭一个打开的对话框。可以使用 open 方法在任何时候打开对话框
'isOpen'	（字符串）如果对话框处于打开状态，则返回 true；否则返回 false
'moveToTop'	（字符串）如果存在多个对话框，则将此对话框移到对话框堆栈的顶层
'widget'	（字符串）返回对话框部件元素，即带有 ui-dialog 类名的元素

返回值

包装集，除了返回如上所述的值的情况

11

理解创建对话框和打开对话框的区别是很重要的。在创建了一个对话框之后，就无需在关闭后需要再次打开的时候再次创建对话框。除非被禁用了，否则对话框在创建后会被自动打开。重新打开已经被关闭的对话框，要调用 `dialog('open')`方法，而不是再次带选项地调用 `dialog()`方法。

和往常一样，对话框实验室可以在文件 chapter11/dialogs/lab.dialogs.html 中找到，如图 11-15 所示。你可以尝试 `dialog()`方法的各个选项。



图 11-15　jQuery UI 对话框实验室可以尝试可用于 jQuery UI 对话框的所有选项

你可以一边通读表 11-16 中的选项列表，一边在实验室进行练习。

表11-16　jQuery UI对话框部件的选项

选　项	描　述	是否在实验室页面中
autoOpen	（布尔）除非设置为false，否则对话框会在创建的时候打开。当设置为false时，可以通过调用dialog('open')来打开对话框	✓
beforeClose	（函数）指定在对话框上创建的函数，用来作为dialogbeforeclose事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	✓
buttons	（对象）指定放置在对话框底部的任何按钮。对象中的每个属性作为按钮的标题，属性的值必须是一个单击按钮时调用的回调函数 调用此处理器时会将函数上下文设置为对话框元素，同时向此处理器传入事件实例参数（按钮组被设置为事件实例的target属性） 如果省略，则不会为对话框创建任何按钮 函数上下文适合与dialog()方法一起使用。例如，在Cancel按钮的回调函数中，下面的代码用来关闭对话框： `$(this).dialog('close');`	✓
close	（函数）指定在对话框上创建的函数，用来作为dialogclose事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	✓
closeOnEscape	（布尔）除非设置为false，否则在对话框获取焦点的时候用户按下Escape键会关闭对话框	✓
closeText	（字符串）用来替换关闭按钮默认值Close的文本值	✓
dialogClass	（字符串）除了jQuery UI向对话框元素添加的类名外，还可以使用此选项来指定添加到对话框元素的以空格分隔的CSS类名。如果省略，则不会添加额外的类名	
drag	（函数）指定在对话框上创建的函数，用来作为dialog drag事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	✓
dragStart	（函数）指定在对话框上创建的函数，用来作为dialog dragStart事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
dragstop	（函数）指定在对话框上创建的函数，用来作为dialog dragStop事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
draggable	（布尔）除非设置为false，否则可以通过单击和拖动对话框的标题栏来移动对话框	✓
focus	（函数）指定在对话框上创建的函数，用来作为dialog dialogfocus事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
height	（数值|字符串）以像素为单位的对话框的高度，或字符串"auto"（默认值），它可以根据对话框自身的内容来决定其高度	✓
hide	（字符串|对象）关闭对话框时使用的特效（参见第9章）。默认为null	✓
maxHeight	（数字）可以调整对话框尺寸的最大高度（以像素为单位）	✓
maxWidth	（数字）可以调整对话框尺寸的最大宽度（以像素为单位）	✓
minHeight	（数字）可以调整对话框尺寸的最小高度（以像素为单位，默认为150）	✓

（续）

选　　项	描　　述	是否在实验室页面中
minWidth	（数字）可以调整对话框尺寸的最小宽度（以像素为单位，默认为150）	✓
modal	（布尔）如果为true，则会在对话框的后面创建一个半透明的“幕帘”，来盖住窗体内容的其余部分，从而阻止用户交互行为 如果省略，则对话框是非模式的	✓
open	（函数）指定在对话框上创建的函数，用来作为dialogopen事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
position	（字符串|数组）指定对话框的初始位置。可以是预定义位置center（默认值）、left、right、top或bottom的其中之一 也可以是一个带有left和top值（以像素为单位）的二维数组[left, top]，或者文本位置例如['left', 'top']	✓
resize	（函数）指定在对话框上创建的函数，用来作为dialog resize事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
resizable	（布尔）除非指定为false，否则对话框可以在各个方向上调整尺寸	✓
resizeStart	（函数）指定在对话框上创建的函数，用来作为dialogresizeStart事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
resizeStop	（函数）指定在对话框上创建的函数，用来作为dialogresizeStop事件的事件处理器。有关这个事件的更多细节（传入处理器的信息）请参见表11-17	
show	（字符串）用来打开对话框的特效。默认情况下，不使用任何特效	✓
stack	（布尔）除非指定为false，否则对话框会在获取焦点的时候移动到任何其他对话框的上面	
title	（字符串）指定显示在对话框边框的标题栏中的文本。默认情况下，将对话框元素的title特性作为标题	✓
width	（数字）以像素为单位的对话框的宽度。如果省略，则使用默认值300	✓
zIndex	（数字）用来覆盖对话框的初始z-index（默认为1000）的值	

使用对话框实验室可以很容易地看到大部分选项的操作，但是确保你了解模式对话框和非模式对话框的区别。

在实验室的控制台，各种事件（随着对话框交互的进行）按照它们被触发的顺序来依次显示。下面来考察可能的事件。

11.8.2　对话框事件

当用户操作我们创建的对话框时，会触发各种自定义的事件，以便于我们向页面中添加钩子（hook）。这让我们有机会在对话框生命周期的适当时间执行操作，甚至影响对话框的操作。

在对话框交互过程中触发的事件显示在表 11-17 中。向每个事件的处理器传递事件实例和自定义对象。函数上下文（也就是事件目标）被设置为对话框元素。

传入处理器的自定义对象取决于事件的类型。

- 对于 dialogdrag、dialogdragStart 和 dialogdragStop 事件，自定义对象包含属性 offset 和 position，它们又包含属性 left 和 top，分别识别对话框相对于页面或者它的偏移父元素的位置。
- 对于 dialogresize、dialogresizeStart 和 dialogresizeStop 事件，自定义对象包含属性 originalPosition、originalSize、position 和 size。position 属性是包含期望的 left 和 top 属性的对象，而 size 属性包含 height 和 width 属性。
- 对于所有其他的事件类型，自定义对象没有属性。

表11-17 jQuery UI对话框的事件

事 件	选 项	描 述
dialogbeforeclose	beforeClose	在对话框将要关闭时触发 返回false来阻止关闭对话框——对于包含错误验证表单的对话框非常方便
dialogclose	close	关闭对话框后触发
drag	drag	当处于拖动操作中移动对话框时，重复地触发
dragStart	dragStart	当通过拖动对话框的标题栏来开始重新定位对话框时触发
dragStop	dragStop	当拖动操作终止时触发
dialogfocus	focus	当对话框获取焦点时触发
dialogopen	open	当打开对话框时触发
resize	resize	当改变对话框尺寸时重复触发
resizeStart	resizeStart	当开始改变对话框尺寸时触发
resizeStop	resizeStop	当结束改变对话框尺寸时触发

在介绍这些事件的一些巧妙用法之前，先来考察一下 jQuery 放置在（参与对话框创建的）元素上的类名。

11.8.3 对话框的类名

和其他部件一样，jQuery UI 为即将进入对话框部件结构中的元素添加了类名，来帮助我们查找这些元素，以及通过 CSS 来为它们添加样式。

对于对话框部件，添加的类名如下所示。

- ui-dialog——添加到<div>元素的类名，这个元素用来包含整个部件，包括内容和边框。

- ui-dialog-titlebar——添加到<div>元素的类名，这个元素用来容纳标题和关闭图标。
- ui-dialog-title——添加到<span>元素的类名，这个元素被包含在标题栏中，用来包裹标题文本。
- ui-dialog-titlebar-close——添加到<a>标签元素的类名，这个元素用来包含标题栏中的“x”（关闭）图标。
- ui-dialog-content——添加到对话框内容元素（调用 dialog()时所操作的包裹元素）的类名。

重要的是要记住，传入事件处理器的元素是对话框内容元素（使用 ui-dialog-content 来标记的元素），而不是生成的用来保持部件的外部容器。

下面来看看几种指定内容的方式，且该内容尚不存在于页面的对话框中。

11.8.4　对话框使用技巧

一般说来，对话框是通过包含在页面标签中的<div>元素来创建的。jQuery UI 获取它的内容，将其从 DOM 中删除，创建作为对话框外框的元素，并设置原始的那个元素作为对话框外框的内容。

但是如果我们想在 dialogopen 事件处理器中通过 Ajax 动态地加载内容，该怎么办呢？实际上使用下面代码能够非常简单地完成这个操作：

```
$('<div>').dialog({
  open: function(){ $(this).load('/url/to/resource'); },
  title: 'A dynamically loaded dialog'
});
```

在这段代码中，我们动态地创建了一个新的<div>元素，然后把它转变为对话框，就好像它是一个已经存在的元素似的。选项指定了对话框的标题，以及一个为 dialogopen 事件准备的回调函数，在回调函数中使用 load()方法来加载内容元素（对话框的内容元素被设置为函数上下文）。

目前看到的场景中，不管是内容已经存在于页面中的情况，还是通过 Ajax 加载内容的情况，内容都是存在于当前页面的 DOM 元素中。如果我们希望对话框的主体内容存在于自己的页面中，该怎么办？

如果对话框内容和页面内容需要以任何方式进行交互，那么使对话框内容和它的父元素位于相同的 DOM 结构中很方便，但是我们也可能希望对话框内容是一个完整独立的页面。最常见的原因可能是因为对话框内容需要自己的样式和脚本，而我们不希望在准备使用对话框的每一个父页面包含这些资源。

怎么来完成这个任务呢？HTML 是否支持使用单独的页面作为另一个页面的一部分呢？当然了…… <iframe>元素！

考虑这段代码：

```
$('<iframe src="content.html" id="testDialog">').dialog({
  title: 'iframe dialog',
  buttons: {
```

```
    Dismiss: function(){ $(this).dialog('close'); }
  }
});
```

这里我们动态地创建了一个`<iframe>`元素，指定它的源和 `id` 特性，然后把它转变为对话框。传入 `dialog()`方法的选项指定了对话框标题和用来关闭对话框的 Dismiss 按钮。是不是非常棒？

现在沾沾自喜还为时过早，显示的对话框出现一个问题。`<iframe>`的滚动条被对话框的边框截掉了一部分，如图 11-16 左半部分所示。我们想要的对话框当然是图右半部分所示的样子。

因为`<iframe>`看起来有点宽，我们可以尝试使用一个 CSS 规则使其变窄，但是令我们失望的是这不起作用。稍微深入探索后发现 `dialog()`方法将一个 CSS 规则 `width:auto` 放置到了`<iframe>`元素上，从而使所有非直接操作`<iframe>`样式的尝试都以失败告终。[1]



图 11-16　我们的喜悦之情被截断的滚动条所破坏，如图左侧所示——如何修正这个问题使其看起来如图右侧所示呢

不过这也没关系。我们可以使用更高级别的样式规则。下面为 `dialog()`调用添加如下选项：

```
open: function(){
  $(this).css('width','95%');
}
```

这会在对话框打开的时候覆盖位于`<iframe>`上的样式。

请记住，通过这种方式创建的对话框也是有缺点的。例如，任何在父页面中创建的按钮和在`<iframe>`中加载的页面的交互将需要跨两个窗体进行交流。

这个示例的源文件可以在 chapter11/dialogs/iframe.dialog.html 中找到。

11.9　小结

哇。这是很长的一章，但是我们也从中学到了很多知识。

[1] 这里涉及 CSS 的优先级规则，一般说来内联样式的优先级最高。详细信息请参考：http://www.w3.org/TR/CSS21/cascade.html#specificity。

我们看到 jQuery UI 是如何在它提供的交互部件和特效（前面几章曾经考察过）的基础之上创建的，以便允许我们创建各种部件，为用户提供直观和易于使用的用户界面。

我们学习了扩展传统 HTML 外观和风格的按钮部件，使其在 jQuery UI 的沙盒中很好地工作。

允许用户输入数字和日期数据类型（传统的实现方式充满了问题）的部件分别由滑动条和日期选择器来提供。自动完成部件完善了数据输入部件，允许用户快速过滤大的数据集。

进度条部件赋予了我们以图形和易于理解的方式向用户展示操作完成状态的百分比。

最后，我们看到了 3 个以不同方式组织内容的部件：选项卡、手风琴和对话框。

将这些部件添加到工具箱中，我们就可以利用它们组织各种界面。不过这只是由 jQuery UI 提供的官方部件集合。我们已经看到，jQuery 可以很方便地进行扩展， jQuery 社区也一直没有闲着。即使没有成千上万，也至少有成百上千个其他的插件存在，等待着我们去探索。可以从 http://plugins.jquery.com/开始你的探索。

11.10　结束语

真不容易！尽管我们在本书中介绍了 jQuery 和 jQuery UI 全部的 API，但是我们无法逐一展示所有 API 在页面上的使用方式。我们介绍的示例都是经过特别挑选的，目的是为了帮助你发现如何使用 jQuery 来解决日常在 Web 应用页面开发中遇到的问题。

jQuery 是一个活力四射的项目。令人吃惊的活跃！唉，对于本书作者来说，如何在编写本书的过程中跟上 jQuery 库快速的开发步伐可是一件苦差事。核心库在不断演变成一个更加有用的资源，几乎每天都会出现很多新的插件。jQuery UI 的开发步伐也在不遗余力地向前推进。

我们强烈希望你能够持续关注 jQuery 社区的发展，并衷心希望本书能够带给你帮助，以更短的时间和更少的代码（比你曾经认为的还要少）编写更好的 Web 应用程序。

我们希望你能健康快乐，希望你能轻松地解决掉所有的 bug！

附　录

JavaScript必知必会

附录内容

- ❑ 哪些 JavaScript 概念对于有效地使用 jQuery 来说很重要
- ❑ JavaScript `Object` 的基础知识
- ❑ 为什么说函数是一等对象
- ❑ 确定（以及控制）`this` 的含义
- ❑ 闭包是什么

jQuery 为 Web 应用带来的最大好处之一就是无需自己编写一大堆脚本代码就能实现大量启用脚本的行为。jQuery 处理了具体的细节，所以我们只须专注于让 Web 应用做它们该做的事情！

在本书的最初几章中，只需要最基本的 JavaScript 技巧就能理解和编写那些示例。在探讨高级主题（比如事件处理、动画、以及 Ajax）的各章中，则必须理解几个基本的 JavaScript 概念，以便能高效地使用 jQuery 库。你可能发现，在 JavaScript 中有很多原以为理所当然（或盲目接受）的事情开始变得更有意义了。

我们不打算在这里探讨所有的 JavaScript 概念——这不是本书的意图。本书旨在帮助读者在尽可能短的时间内快速有效地运用 jQuery。为此，附录将会专注于基本概念，以便在 Web 应用程序中最有效地使用 jQuery。

这些概念中最重要的就是：函数是 JavaScript 中的一等对象，JavaScript 定义和处理函数方式也体现出这一点。这意味着什么呢？为了理解函数是对象，先抛开一等对象的说法，我们首先必须确保要能理解 JavaScript 对象的全部含义。下面就来深入探讨吧。

A.1　JavaScript对象的基本原理

大多数面向对象（简称 OO）语言都定义了某种基本的 `Object` 对象类型，其他所有的对象都源于这个对象类型。在 JavaScript 中，基本的 `Object` 对象确实是作为所有其他对象的基础，但是这种比较也仅此而已。从基本层面上看，JavaScript 的 `Object` 对象与其他大部分的 OO 语言所定义的基本对象毫无相通之处。

初看起来，JavaScript 的 `Object` 对象可能是平淡无奇，甚至是令人感到乏味的。对象一旦被创

建后,它不保存任何数据并且几乎没有什么语义。但是这些有限的语义的确又给予了它很大的潜力。

下面就来一探究竟吧。

A.1.1　对象从何而来

新的对象由 new 操作符和与其相配的 Object 构造器来产生。创建一个对象非常简单，如下所示：

```
var shinyAndNew = new Object();
```

它还可以更简单（很快就会看到），不过目前这就可以了。

但这个新对象能做些什么呢？其中似乎什么也没有，没有信息，没有复杂的语义，什么也没有。这新出炉的闪亮对象一点也不吸引人，直到我们开始向其添加东西——称为*属性*的东西。

A.1.2　对象的属性

就像服务器端对象那样，JavaScript 对象也可以包含数据和方法（噢……有点像，不过对我们来说这些内容有点超前了）。不同于那些服务器端的对象，这些元素不是为对象预先声明的，我们可以在需要的时候动态地创建它们。

来看看下面的代码片段：

```
var ride = new Object();
ride.make = 'Yamaha';
ride.model = 'V-Star Silverado 1100';
ride.year = 2005;
ride.purchased = new Date(2005,3,12);
```

在这里我们创建了一个新的 Object 实例，并且把它赋值给 ride 变量。然后用一些不同类型的*属性*来填充这个变量：两个字符串、一个数字以及一个 Date 类型的实例。

不需要在赋值前声明这些属性，这些属性只不过是通过赋值而产生的。这是给予我们高度灵活性的强大“符咒”。不过，在你得意忘形之前，别忘了灵活性总是要付出代价的！

例如，假设在启用脚本的 HTML 页面的后续代码中，我们想要改变购买日期：

```
ride.purchased = new Date(2005,2,1);
```

没问题……除非我们不小心打错字了，比如

```
ride.purchased = new Date(2005,2,1);
```

没有编译器会警告我们犯了个错误。我们高高兴兴地创建了名为 purchased 的新属性，可随后我们就会很纳闷，在引用拼写正确的 purchased 属性时为什么新的日期没有**生效**呢？

能力越大，责任越重（似曾听到过？），因此打字要仔细！

注意　在处理这些问题的时候，JavaScript 的调试工具（比如 Firefox 的 Firebug）可以说是“救生员”。因为类似这样的打字错误通常不会导致 JavaScript 错误，而依赖 JavaScript 控制台或错误对话框则通常不起作用。

从这个例子中，我们知道了 JavaScript Object 的实例（从现在开始简称为对象）就是一组属性集，每个属性都由名称和值构成。属性的名称是字符串，属性值可以是任何 JavaScript 对象，可以是 Number、String、Date、Array、基本的 Object，也可以是任何其他的 JavaScript 对象类型（也包括函数，我们很快就会看到）。

这意味着 Object 实例的主要就是用作为容器，包含其他对象的已命名集合。这可能会让你想起其他语言中的概念：例如 Java 中的映射，或其他语言中的字典或散列。

属性不局限于类似于 String 或 Number 的类型。一个对象属性可以是另一个 Object 实例，这个实例又包含其自己的属性集，而属性集中也可以包含拥有属性的对象，以此类推。只要对我们塑造的数据模型有意义，就可以嵌套至任何层次。

假设给 ride 实例添加一个新的属性以保存车辆的所有者信息。这个属性是另一个 JavaScript 对象，它包含了一些属性，如所有者姓名和职业：

```
var owner = new Object();
owner.name = 'Spike Spiegel';
owner.occupation = 'bounty hunter';
ride.owner = owner;
```

为了访问嵌套的属性，我们编写如下代码：

```
var ownerName = ride.owner.name;
```

可利用的嵌套层次是没有限制的（只要不超出常识的限度）。当完成此对象后（目前为止），对象的层次结构如图 A-1 所示。

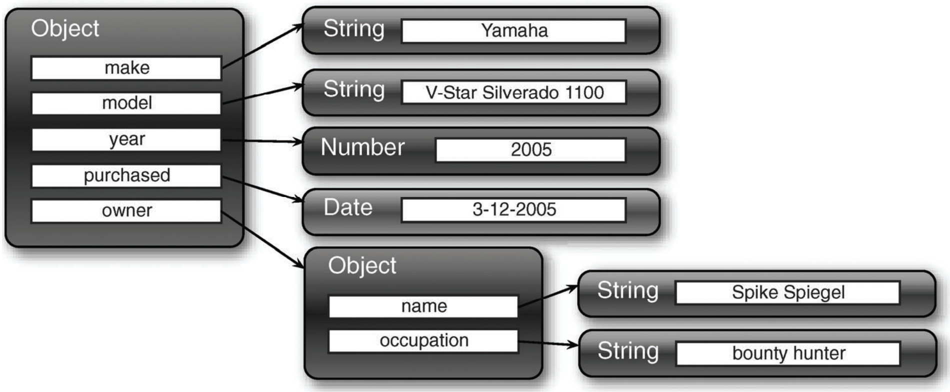


图 A-1　对象的层次结构显示了 Object 是其他 Object 或 JavaScript 内置类型的已命名引用的容器

注意该图中的每个值都是一个不同的 JavaScript 实例。

注意　在这些代码片段中，出于解释的目的而创建的所有中间变量都是可以省略的（比如 owner）。很快我们就会看到使用更加高效和简洁的方式来声明对象及其属性的代码。

目前为止，我们使用点操作符（英文的句号字符）来引用对象的属性。但是事实证明，有一个更加通用的操作符（点操作符的同义词）来执行属性引用。

例如，如果有一个名为 color.scheme 的属性，那会怎么样呢？你有没有注意到名称中间的点号？它会破坏整个操作，因为 JavaScript 解释器会试图将 scheme 作为 color 的嵌套属性来查找。

“哦，只要不那样做就行！”你会这样说。但是如果出现空格符呢？如果出现可能会被误认为是分隔符而不是名称一部分的其他字符，该怎么办呢？最重要的是，如果我们不知道属性的名称是什么，只知道它是另一个变量的值或某个表达式的求值结果，又该怎么办呢？

在所有的这些情况下，点操作符都是不合适的，因此必须使用更加通用的表示法来访问属性。通用的属性引用操作符的格式是

```
object[propertyNameExpression]
```

其中 propertyNameExpression 是 JavaScript 表达式，其求值的结果作为要引用的属性名称的字符串。例如，下面的 3 个引用都是等价的：

```
ride.make
ride['make']
ride['m'+'a'+'k'+'e']
```

也等价于下面这个引用：

```
var p = 'make';
ride[p];
```

对于其名称并非有效的 JavaScript 标识符的属性来说，使用通用的引用操作符是引用这种属性的唯一方法，例如，

```
ride["a property name that's rather odd!"]
```

包含了对于 JavaScript 标识符来说不合法的字符，或者属性的名称是其他变量的值。

通过 new 操作符来创建新的实例，并且利用独立的赋值语句来为每个属性赋值从而建立对象，是一件繁琐的事情。在下一节中，我们将会探讨声明对象及其属性的更加紧凑和易读的表示法。

A.1.3　对象字面值

在上一节中，我们创建了一个对摩托车属性进行建模的对象，并将其赋值给 ride 变量。为了完成个操作，我们使用了两个 new 操作符、一个名为 owner 的中间变量以及一些赋值语句。这样做既枯燥乏味，又冗长易错），并且难以在快速检查代码的时候把握对象的结构。

幸运的是，我们可以使用更紧凑和更易于阅读的表示法。考虑如下语句：

```
var ride = {
  make: 'Yamaha',
  model: 'V-Star Silverado 1100',
  year: 2005,
  purchased: new Date(2005,3,12),
  owner: {
```

```
    name: 'Spike Spiegel',
    occupation: 'bounty hunter'
  }
};
```

这个代码片段使用对象字面值来创建 ride 对象，与上一节中用赋值语句创建的 ride 对象相同。

这种表示法被称为 JSON（JavaScript Object Notation，JavaScript 对象表示法[①]），大多数页面开发者对 JSON 的偏爱有加，不喜欢通过多个赋值语句来建立对象的方式。JSON 的结构简单：对象由一对花括号表示，在其中列出用逗号分隔的多个属性。每个属性通过以冒号分隔的名称和值来表示。

注意　从技术上而言，JSON 无法表示日期值，主要是因为 JavaScript 本身没有任何一种日期字面值。当在脚本中使用 JSON 时，通常会使用 Date 构造器来表示日期值，如前面示例所示。如果用作交换格式使用时，日期常常被表示为包含 ISO 8601 格式的字符串或表示日期值的数字（Date.getTime()返回的毫秒数）。

同样也要注意，当把 JSON 用作交换格式使用时，必须遵守一些严格的规则，比如引用属性的名称[②]。访问 http://www.json.org 或者 RFC 4627（http://www.ietf.org/rfc/rfc4627.txt）以了解更多信息。

从 owner 属性的声明可见，对象的声明可以嵌套。

顺便说一下，也可以在 JSON 中通过在方括号内放置以逗号分隔的元素列表来表示数组，如下所示：

```
var someValues = [2,3,5,7,11,13,17,19,23,29,31,37];
```

和我们在本节的示例中看到的一样，对象引用通常存储在变量或其他对象的属性中。下面就来看看关于后者的特殊案例。

A.1.4　作为窗体属性的对象

到目前为止，我们已经看到了两种保存 JavaScript 对象的方式：变量和属性。这两种保存引用的方式使用不同的表示法，如下面的代码片段所示：

```
var aVariable =
  'Before I teamed up with you, I led quite a normal life.';

someObject.aProperty =
  'You move that line as you see fit for yourself.';
```

---

① 更多信息，请访问 http://www.json.org/。

② 作为数据交换格式时，JSON 中属性的名称必须由双引号来引用，这和作为 JavaScript 对象字面值的属性可以由单引号、双引号，或者干脆不用引号（只要属性名称不包含空格等特殊字符）来引用不同。

这两个语句分别把（通过字面值创建的）`String` 实例赋值给第一个语句中的变量及第二个语句中的对象属性。（如果能够找出这两个晦涩引文的出处，那么荣誉就归你了；不能用 Google 作弊！本章的前面部分留有线索[①]。）

但是这两个语句真的执行不同的操作吗？事实证明不是的！

如果在顶层（任何内部的函数主体之外）中使用 `var` 关键词时，那只不过是对程序员友好的表示法，用来引用预定义在 JavaScript `window` 对象上的属性。任何在顶层作用域中生成的引用都隐式地创建在 `window` 实例中。这意味着下面所有的语句，如果是在顶层中（也就是在函数的作用域之外）生成的，那么它们都是等价的：

```
var foo = bar;
```

和

```
window.foo = bar;
```

以及

```
foo = bar;
```

不管使用的是哪种表示法，都会创建一个名为 `foo` 的 `window` 属性（如果 `foo` 属性尚未存在），并且将 `bar` 赋值给 `foo`。还要注意，因为 `bar` 是非限定的，所以将其假定为 `window` 的一个属性。

把顶层作用域认为是 `window` 作用域，这可能不会让我们陷入概念上的烦恼，因为任何位于顶层的未限定的引用都被假定为 `window` 的属性。当深入研究函数主体时，作用域规则会变得更加复杂（事实上会极其复杂），不过我们很快就会解决这个问题。

以上涵盖了 JavaScript `Object` 综述的大部分内容。从上述讨论中总结出来的重要概念有：

- JavaScript 对象是属性的无序集合；
- 属性由名称和值组成；
- 可以使用字面值来声明对象；
- 顶层变量是 `window` 的属性。

现在来讨论一下，当我们谈到 JavaScript 函数是一等对象时，意味着什么？

A.2　作为一等公民的函数

在许多传统的 OO 语言中，对象可以包含数据，还可以拥有方法。在这些语言中，数据和方法通常是不同的概念，而 JavaScript 却走了一条不同的道路。

JavaScript 中的函数也被认为是对象，与定义在 JavaScript 中任何其他的对象类型一样，比如 `String`、`Number` 或 `Date`。和其他对象一样，函数也是通过 JavaScript 构造器来定义的（在这种情况下是 `Function`），可以对函数进行如下操作：

- 把函数赋值给变量；

① 这两处引用是 `ride.owner` 提到的电影《赏金猎人》中的台词。

- 将函数指定为一个对象的属性；
- 把函数作为参数传递；
- 把函数作为函数结果返回；
- 使用字面值来创建函数。

因为在 JavaScript 语言中对待函数的方式与对待其他对象的方式相同，所以我们说函数是一等对象。

但是你可能心里会想，函数与其他对象类型（例如 `String` 或 `Number`）从根本上是不同，因为函数不仅拥有值（在 `Function` 实例中，值是函数体），而且还拥有*名称*。

恩，还早着呢！[①]

A.2.1　名称中包含了什么

大部分 JavaScript 程序员都有一种错误的假设，认为函数是已命名的实体。事实并非如此，如果你就是这种程序员，那么你已经被绝地武士的控心术[②]愚弄了。与对象的其他实例（例如 `String`、`Date` 或 `Number`）一样，只有在把函数赋值给变量、属性或参数的时候，函数才能被引用。

下面思考类型为 `Number` 的对象。我们经常通过字面值表示法来表示 `Number` 实例，例如下面的语句

```
213;
```

是完全有效的，但是它也是完全无用的。`Number` 实例用处不大，除非将其赋值给属性或对象，或者将其绑定到参数名称上。否则，我们无法引用散落在内存中的实例。

同样的规则也适用于 `Function` 对象的实例。

“但，但是……”你可能会说，“那下面的代码呢？”

```
function doSomethingWonderful() {
  alert('does something wonderful');
}
```

“那不是创建了名为 `doSomethingWonderful` 的函数吗？”

不，不是的。尽管这种表示法可能看起来很熟悉，而且被广泛用来创建*顶层*函数，但它与通过 `var` 来创建 `window` 属性使用的是相同的语法。`function` 关键字自动创建一个 `Function` 实例并将其赋值给使用函数“名称”创建的 `window` 属性（前面称其为*绝地武士的控心术*），如下所示：

```
doSomethingWonderful = function() {
  alert('does something wonderful');
}
```

如果这看起来有点奇怪，考虑另一个使用完全相同形式的语句，但这次使用 `Number` 的字面值：

```
aWonderfulNumber = 213;
```

① 也就是说上面对函数名称和值的理解不完全正确，还有很多知识需要学习。

② Jedi mind trick 是《星球大战》里绝地武士的一个招式，用于控制对方的思维。

这个语句不足为不奇，它与把函数赋值给顶层变量（window 属性）的语句如出一辙：使用函数字面值来创建 Function 实例，然后将函数赋值给变量 doSomethingWonderful，与使用 Number 字面量 123 将 Number 实例赋值给变量 aWonderfulNumber 的方式完全相同。

如果你从来没有看到过函数字面值的语法，可能会觉得它有点奇怪。它是由关键字 function 与紧接着的被圆括号所包含的参数列表，以及随后的函数主体所组成的。

如果我们声明顶层的已命名函数时，就会同时创建一个 Function 实例，并将其赋值给 window 的一个属性，该属性使用所谓的函数名称自动创建。Function 实例不再拥有名称，只是拥有 Number 字面值或 String 字面值。图 A-2 说明了这个概念。

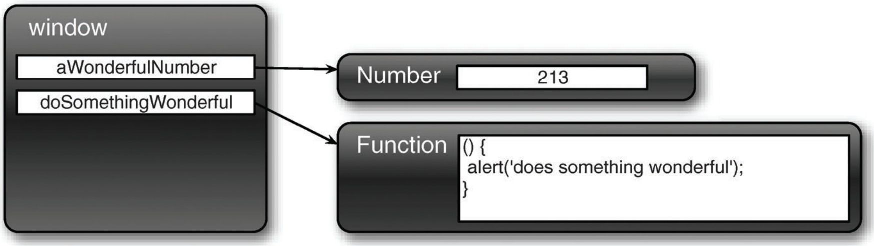


图 A-2　Function 实例是无名称的对象，就像 Number 213 或任何其他的 JavaScript 值一样。只能通过生成引用来对其进行命名

记住，在 HTML 页面中创建了顶层变量时，会将变量创建为 window 实例的属性。因此，下面的语句都是等价的：

```
function hello(){ alert('Hi there!'); }
hello = function(){ alert('Hi there!'); }
window.hello = function(){ alert('Hi there!'); }
```

尽管这看起来很像语法"魔术"，但是理解这一点很重要：和其他对象类型的实例一样，Function 实例可以赋值给变量、属性或参数的值。并且就像其他的那些对象类型，无名称无实体的实例毫无用处，只有将它们赋值给变量、属性或参数，这样才能引用它们。

Gecko 浏览器和函数名称

基于 Gecko 布局引擎的浏览器（如 Firefox 以及 Camino），把使用顶层语法定义的函数名称存储在函数实例中名为 name 的非标准属性中。尽管对于一般的公共开发而言这可能没有什么用处（尤其是考虑到它受限于基于 Gecko 的浏览器），但是对于浏览器插件和调试器的开发者来说这是非常有价值的。

我们已经看到了将函数赋值给变量、属性的示例，将它作为参数传递会是什么情况呢。下面来看看这么做的原因和方式。

A.2.2　作为回调的函数

当我们的代码遵循井然有序的同步流时，顶层函数当然不错，但是 HTML 页面的本质是，

一旦加载之后就完全不是同步的。无论我们是在处理事件、创建计时器，还是发起 Ajax 请求，Web 页面代码的本质是异步的。在异步编程中最流行的概念是*回调函数*。

以计时器为例，可以通过向 `window.setTimeout()`方法传递合适的持续时间值来触发计时器（假设 5 秒之后触发计时器）。但是这个方法如何才能让我们知道计时器什么时候过期，让我们不用空等而做其他事情？这个方法通过调用由我们提供的函数来完成这一任务。

思考下面的代码：

```
function hello() { alert('Hi there!'); }

setTimeout(hello,5000);
```

我们声明了一个 `hello` 函数并设置在 5 秒钟（第二个参数表示 5000 毫秒）之后触发计时器。我们向 `setTimeout()`方法传递的第一个参数是一个函数的引用。传递函数作为参数与传递任何其他值没有什么不同——就像传递 `Number` 作为第二个参数那样。

当计时器过期时，`hello` 函数会被调用。因为在代码中 `setTimeout()`方法*回调*了一个函数，所以该函数被称为*回调函数*。

大部分高级 JavaScript 程序员可能会觉得这段代码示例很幼稚，因为没有必要创建 `hello` 名称。除非要在页面的其他地方调用此函数，否则没有必要创建 `window` 的属性 `hello` 来暂时存储 `Function` 实例，以便将 `hello` 作为回调参数来传递。

编写这个片段更简洁的方式是：

```
setTimeout(function() { alert('Hi there!'); },5000);
```

这个代码在参数列表中直接使用函数字面值，没有生成不必要的名称。这是我们在 jQuery 代码中经常看到的习惯用法（当不必把函数实例赋值给顶层属性时）。

目前为止，示例中所创建的函数要么是顶层函数（也就是顶层的 `window` 属性），要么是在函数调用中被赋值给参数。我们也可以将 `Function` 实例赋值给对象的属性，此时事情才真正变得有趣起来。继续阅读吧……

A.2.3　`this`到底是什么

OO 语言自动提供了从方法中引用对象当前实例的办法。在 Java 和 C++这样的语言中，`this` 参数指向当前实例。在 JavaScript 中，也存在个类似的概念，甚至还使用了相同的 `this` 关键词，同样也提供方法来访问与函数相关联的对象。但是 OO 程序员要注意！JavaScript 实现中的 `this` 和其他 OO 语言中的 `this` 存在微妙的差异，这种差异体现在几个重要的方面。

在基于类的 OO 语言中，`this` 指针通常引用类的实例（方法在此类中声明）。在 JavaScript 中函数是一等对象，它们不被声明为任何东西的一部分，而 `this` 所引用的对象（称为*函数上下文*）并不是由声明函数的方式决定的，而是由调用函数的方式决定的。

这意味着，同样的函数可以有不同的上下文，取决于调用函数的方式。初看起来这可能有点奇怪，但是它相当有用。

在默认情况下，函数调用的上下文（`this`）是对象，其属性包含用于调用该函数的引用。

作为演示，回顾一下摩托车示例，按如下所示修改对象的创建过程（添加的内容以粗体突出显示）：

```
var ride = {
  make: 'Yamaha',
  model: 'V-Star Silverado 1100',
  year: 2005,
  purchased: new Date(2005,3,12),
  owner: {name: 'Spike Spiegel',occupation: 'bounty hunter'},
  whatAmI: function() {
    return this.year+' '+this.make+' '+this.model;
  }
};
```

我们向原始示例代码添加了引用 Function 实例的、名为 whatAmI 的属性。新的对象层次结构（其中 Function 实例被赋值给名为 whatAmI 的属性）如图 A-3 所示。

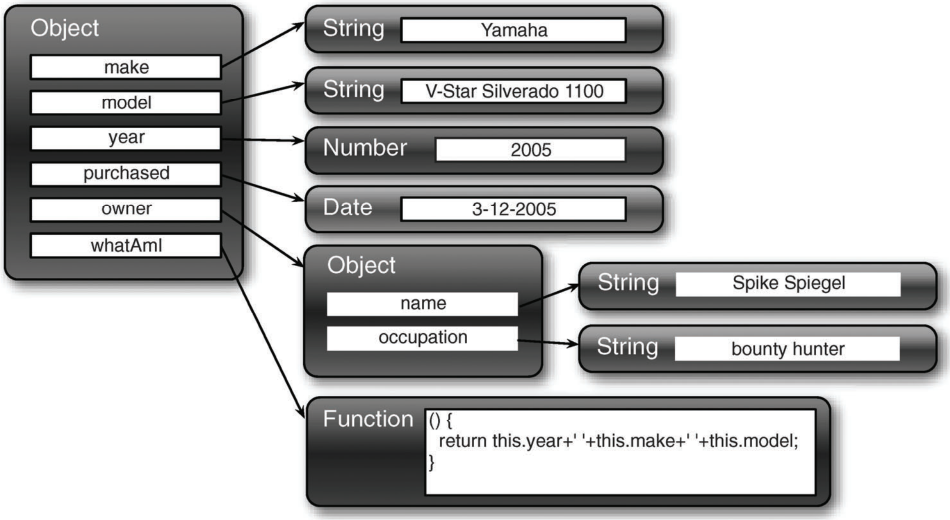


图 A-3　这个模型清楚地显示了函数不是 Object 对象的一部分，而只是被名为 whatAmI 的 Object 属性所引用

当通过属性引用来调用函数时，就像这样，

```
var bike = ride.whatAmI();
```

函数上下文（this 引用）被设置为 ride 所指向的对象实例。作为结果，变量 bike 被设置为字符串 2005 Yamaha V-Star Silverado 1100，因为该函数是通过 this 调用从而获取对象属性的。

对于顶层函数来说也是一样。请记住：顶层函数是 window 的属性，因此当将其作为顶层函数来调用时，其函数上下文就是 window 对象。

尽管这可能是常见的隐式行为，但是 JavaScript 提供的办法可以显式地控制函数上下文所要

使用的内容。通过 Function 的方法 call()或 apply()来调用函数，就可以将函数上下文设置为任何想要的内容。

是的，作为一等对象，函数甚至拥有由 Function 构造器定义的那些方法。

使用 call()方法来调用函数，指定第一个参数作为函数上下文的对象，而其余参数成为被调用函数的参数。也就是说，call()的第二个参数成为了被调用函数的第一个参数，依此类推。apply()方法的工作方式与此类似，不同的是，它的第二个参数应该成为被调用函数参数的对象数组。

困惑了？该是介绍一个综合示例的时候了。思考代码清单 A-1 中的代码（可以从本书源代码 appendix/function.context.html 中找到）。

代码清单 A-1　函数上下文的值取决于调用函数的方式

```
<html>
  <head>
    <title>Function Context Example</title>
    <script>
      var o1 = {handle:'o1'};                    ❶
      var o2 = {handle:'o2'};
      var o3 = {handle:'o3'};
      window.handle = 'window';

      function whoAmI() {                        ❷
        return this.handle;
      }

      o1.identifyMe = whoAmI;                    ❸

      alert(whoAmI());                           ❹
      alert(o1.identifyMe());                    ❺
      alert(whoAmI.call(o2));                    ❻
      alert(whoAmI.apply(o3));                   ❼

    </script>
  </head>
  <body>
  </body>
</html>
```

在这个示例中，我们定义了 3 个简单的对象，每个对象都有 handle 属性，以便在给定引用的时候能够轻松地识别对象❶。我们也向 window 实例添加了 handle 属性以便易于标识。

然后定义了顶层函数，无论什么对象作为其函数上下文，它都会返回 handle 属性值❷，将相同的函数实例赋值给对象 o1 的 identifyMe 属性❸。可以这样认为，这在 o1 上创建了一个 identifyMe 方法，注意到这一点很重要：函数的声明是独立于对象 o1 之外的。

最后，我们弹出 4 个警告框，每个警告框都使用不同的机制来调用相同的函数实例。在浏览器中加载这个示例时，这 4 个警告框依次显示在图 A-4 中。

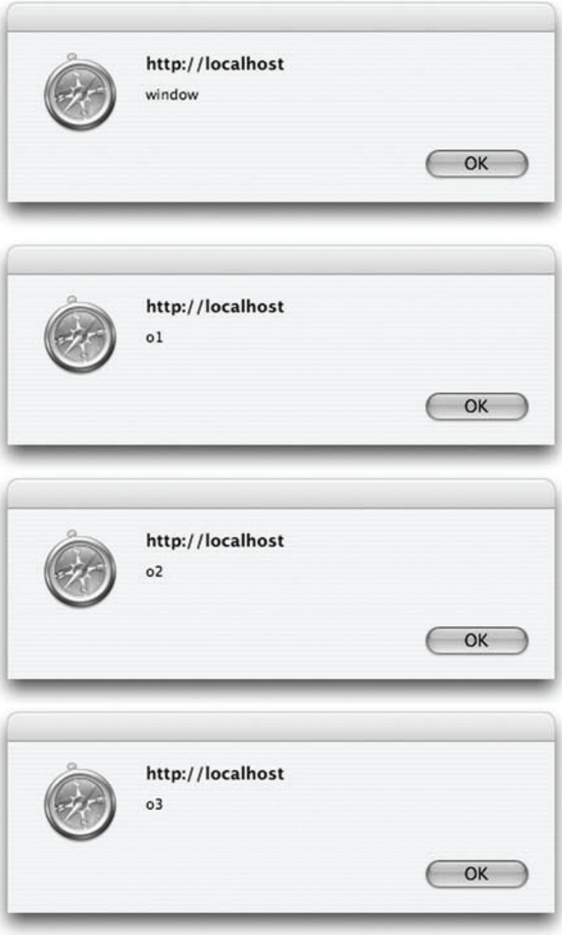


图 A-4　作为函数上下文的对象随着调用函数方式的不同而改变

这个警告框序列说明了以下几个问题。

- 当直接把函数作为顶层函数来调用时，函数上下文是 `window` 实例❹。
- 当把函数作为对象（在这种情况下是 `o1`）的属性来调用时，该对象就成为函数调用的函数上下文❺。可以说函数充当了该对象的*方法*——就像在 OO 语言中那样。但是注意不要为这个类比暗自高兴。如果你不够仔细的话就很可能会误入歧途，就像这个示例的其余部分显示的那样。
- 使用 `Function` 的 `call()`方法将把函数上下文设置为传入 `call()`的第一个参数的任何对象——在这种情况下是 `o2`❻。在这个示例中，函数表现得像是 `o2` 的方法，尽管它和 `o2` 之间没有任何联系，甚至连 `o2` 的属性都不是。
- 就像使用 `call()`那样，可以使用 `Function` 的 `apply()`方法将函数上下文设置为作为第

一个参数传递的任何对象❼。只有在把参数传入函数的时候（为了清晰起见，在本示例中没有进行这个操作），`call()`和`apply()`这两个方法的差异才变得明显起来。

这个示例页面清楚地演示了：函数上下文是基于每个调用来决定的，可以使用任何对象作为函数上下文来调用单个函数。因此，“函数是对象的方法”这个说法绝对是不正确的。下面的表述准确多了：

> 当对象 o 充当函数 f 的调用函数上下文时，函数 f 就充当了对象 o 的方法。

为了更进一步说明这个概念，考虑给示例添加如下语句所带来的效果：

```
alert(o1.identifyMe.call(o3));
```

尽管我们将函数作为 `o1` 的属性来引用，但是这个调用的函数上下文是 `o3`，这进一步强调了：函数上下文取决于调用函数的方式，而不是声明函数的方式。

当使用带有回调函数的 jQuery 命令和函数时，这将会被证明是一个重要概念。我们在 2.3.3 节中就看到过这个概念的实际操作（即使你当时没有意识到这个概念），在那里我们给$的`filter()`方法提供了一个回调函数，把包装集中的每个元素作为其函数上下文来依次顺序地调用函数。

既然已经理解了函数如何充当对象的方法，下面就来把注意力转移到另一个高级的函数话题，在有效利用 jQuery 方面，它将扮演一个重要角色——闭包。

A.2.4 闭包

对于拥有传统的 OO 语言或者过程式编程背景的页面开发者来说，闭包是个理解起来常常有点奇怪的概念。然而，对于拥有函数编程背景的程序员来说，闭包是个熟悉和惬意的概念。先来回答初学者的问题：什么是闭包？

简单地表述如下：闭包就是 `Function` 实例，它结合了来自环境的（函数执行所必需的）局部变量。

在声明函数时，可以在声明函数时引用函数作用域内任何变量。对于任何技术背景的任何开发者来说，这是预料之中理所当然的事情。但是使用闭包时，即使在函数声明**之后**，已经超出函数作用域（也就是关闭了函数声明）的情况下，该函数仍然带有这些变量。

对于编写有效的 JavaScript 来说，在回调函数运行时，引用声明回调函数时的局部变量是一个不可或缺的工具。再次使用计时器，来看一下代码清单 A-2 中用于演示的示例（文件 appendixA/closure.html）。

代码清单 A-2 闭包允许访问函数声明的作用域

```
<html>
  <head>
    <title>Closure Example</title>
    <script type="text/javascript"
            src="../scripts/jquery-1.2.js"></script>
```

```
    <script>
      $(function(){
        var local = 1;                                              ❶
        window.setInterval(function(){                                          ❷
          $('#display')
            .append('<div>At '+new Date()+' local='+local+'</div>');
          local++;                                           ❸
        },3000);
      });
    </script>
  </head>
  <body>
    <div id="display"></div>                                  ❹
  </body>
</html>
```

在这个示例中，我们定义了在 DOM 加载后触发的就绪处理器。在这个处理器中声明了一个名为 `local` 的局部变量❶，把数字值 1 赋值给它。然后使用 `window.setInterval()`方法创建了一个每隔 3 秒就触发的计时器❷。我们指定了内联函数作为计时器的回调函数，它引用 `local` 变量，会显示当前时间以及 `local` 的值（通过把<div>元素写入到在页面主体内定义的、名为 `display` 的元素中来显示结果❹）。作为回调函数的一部分，`local` 变量的值也被增加了❸。

在运行这个示例之前，如果我们不熟悉闭包，可能会去查看这段代码，从而发现一些问题。我们可能会猜测，因为回调会在页面加载后 3 秒触发（在就绪处理器执行完毕很久以后），所以 `local` 的值在回调函数的执行期间是未定义的。毕竟，`local` 声明所在的块在就绪处理器执行完毕时超出了作用域，对吧？

在页面加载之后并让其运行一小段时间，就可以看到如图 4-5 所示的画面。

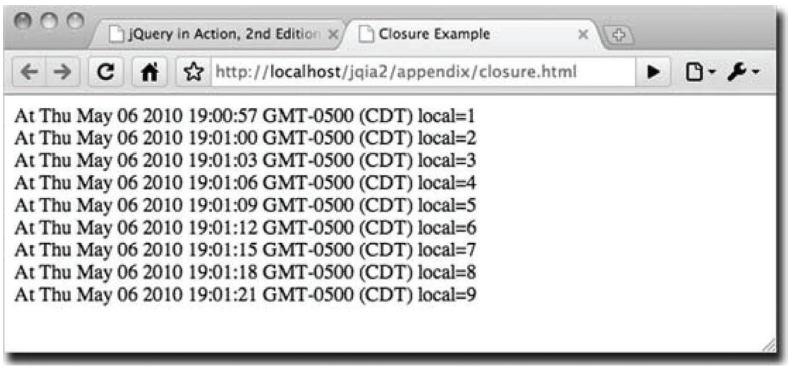


图 A-5　闭包允许回调访问环境中的值，即使该环境已经超出了作用域

它可以正常运行！但它是如何运行的呢？

当就绪处理器执行完毕时，`local` 声明所在的块超出了作用域（这虽然是正确的），但是函数声明所创建的闭包（包括 `local` 变量）在函数的生命周期内都保持在作用域中。

注意　你可能已经注意到，在 JavaScript 中的闭包，其创建方式都是隐式的，而不像其他一些支持闭包的语言那样需要显式的语法。这是一把双刃剑，一方面使得创建闭包很容易（不管你是有意的还是无意的！），但另一方面这使得很难在代码中发现闭包。

无意创建的闭包可能会带来意料之外的结果。例如，循环引用可能导致内存泄露。内存泄露的典型示例就是创建向后引用闭包中变量的 DOM 元素，这会阻止那些变量的回收。

闭包的另一个重要特征是：函数上下文绝不会被包含为闭包的一部分。例如，下面代码不会按照我们期望的方式来执行：

```
...
this.id = 'someID';
$('*').each(function(){
  alert(this.id);
});
```

记住，每个函数调用都有其函数上下文，因此在上述代码中，传递给 `each()`的函数上下文在回调函数内是来自于 jQuery 包装集的元素，而不是被设置为`'someID'`的外部函数属性。对回调函数的每个调用都会依次弹出警告框，用来显示包装集中各个元素的 `id`。

如果需要访问在外部函数中作为函数上下文的对象，可以采用通常的习惯用法：在局部变量中创建 `this` 引用的副本，这个副本将会被包含在闭包中。考虑对示例做如下改变：

```
this.id = 'someID';
var outer = this;
$('*').each(function(){
  alert(outer.id);
});
```

局部变量 `outer` 被赋值为外部函数的函数上下文的引用，从而成为闭包的一部分，可以在回调函数中访问此变量。改变后的代码现在会弹出显示字符串`'someID'`的警告框，包装集中有多少个元素就弹出多少次。

当使用 jQuery 命令（这些命令利用异步的回调函数）来创建优雅的代码时，我们将会发现闭包是必不可少的，尤其是在 Ajax 请求和事件处理领域。

A.3　小结

JavaScript 是在 Web 上被广泛使用的语言，但是许多编写 JavaScript 的页面开发者常常没有深入地使用它。在本附录中，我们介绍了这门语言的一些较为深入的方面，要想在页面上高效地使用 jQuery 就必须了解这些内容。

我们看到了 JavaScript 的 `Object` 主要是作为其他对象的容器而存在。如果你拥有 OO 语言背景，那么把 `Object` 实例看成一个“名称/值”对的无序集合，这可能与你之前所认为的对象大不相同。但即使是在编写不太复杂的 JavaScript 时，这也是一个需要掌握的重要概念。

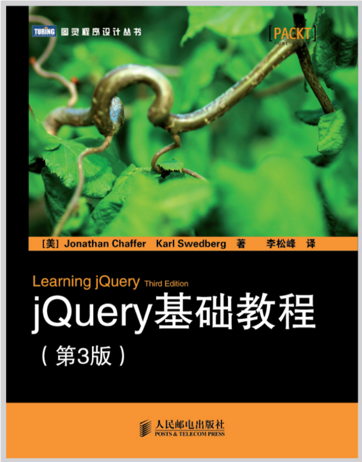
函数在 JavaScript 中是一等公民，能够以类似于其他对象类型的方式来进行声明和引用。可以使用字面值表示法来声明函数，将它们保存在变量和对象属性中，甚至还可以将它们作为回调

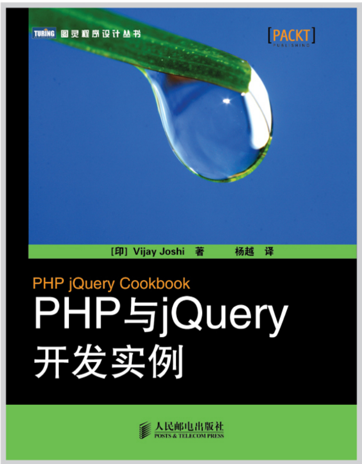
函数，将它们作为参数来传入其他函数。

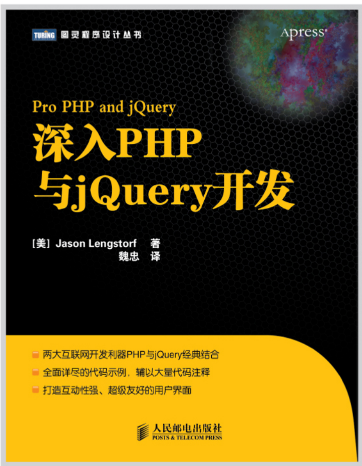
函数上下文这个术语描述的是在函数调用中 `this` 指针所引用的对象。尽管通过将对象设置为函数上下文，就可以像操作对象方法那样来操作函数，但是函数并没有被声明为任何单个对象的方法。调用方式（可由调用者显式地控制）决定了调用的函数上下文。

最后，我们看到了函数声明及其环境形成闭包的方式。在后续调用函数时，这个闭包允许函数访问成为闭包一部分的局部变量。

牢牢地掌握这些概念，我们已经准备好去面对挑战，直面在页面上使用 jQuery 来编写高效的 JavaScript 时会遇到的挑战。


图灵程序设计丛书
PACKT
[美] Jonathan Chaffer Karl Swedberg 著 李松峰 译
Learning jQuery Third Edition
jQuery基础教程
（第3版）
人民邮电出版社
POSTS & TELECOM PRESS


图灵程序设计丛书
PACKT
[印] Vijay Joshi 著 杨越 译
PHP jQuery Cookbook
PHP与jQuery
开发实例
人民邮电出版社
POSTS & TELECOM PRESS


图灵程序设计丛书
Apress
Pro PHP and jQuery
深入PHP
与jQuery开发
[美] Jason Lengstorf 著
魏忠 译
两大互联网开发利器PHP与jQuery经典结合
全面详尽的代码示例，辅以大量代码注释
打造互动性强、超级友好的用户界面
人民邮电出版社
POSTS & TELECOM PRESS

"这本书带我走进了jQuery的神奇世界，让我见识到jQuery的优雅、简洁。我非常享受这段学习过程。"
——Jan Van Ryswyck，elegantcode.com

"随着你在日常开发中对jQuery的了解越来越多，此书也可以作为很好的参考手册。"
——David Hayden， 微软MVP C#，Codebetter.com

jQuery in Action Second Edition

jQuery实战（第2版）

jQuery是目前应用最广的优秀开源JavaScript/Ajax框架之一，已经成为微软ASP.NET、Visual Studio和诺基亚Web Run Time等主流开发平台的组成部分。借助jQuery的魔力，数十行JavaScript代码可以神奇地压缩成区区几行，这让Web开发人员一瞬间就深深地迷恋上这个方便快捷、功能完备的利器。

本书是带领你自如驾驭jQuery的导航者，替你肃清学习和开发路上的各种障碍。在这里，你不仅能深入学习jQuery的各种特性和技巧，还能领略到它的内部工作机制、插件体系结构和背后的各种策略和理论，学会怎样与其他工具和框架交互。这一版基于jQuery1.4讨论了新版本所增加和修改的特性，并增加了3章的篇幅来介绍jQuery UI。

有了jQuery和这本书，你不需要再费心劳力地纠缠于各种高深复杂的JavaScript技巧，只使用CSS、XHTML以及普通的 JavaScript知识，就能直接操作页面元素，实现更快速更高效的Web开发。

- 畅销书升级版，涵盖jQuery1.4和jQuery UI 1.8
- jQuery开发团队核心成员倾力打造
- 掌握Web开发利器的必修宝典

MANNING

图灵社区：www.ituring.com.cn
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机/网络开发

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-27457-1

ISBN 978-7-115-27457-1
定价：69.00元

欢迎加入

图灵社区

最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版的梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要你有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

最直接的读者交流平台

在图灵社区，你可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。

ituring.com.cn